ENSG Géomatique

ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

Calcul de la position d'un récepteur GNSS "à la main"

Clément Fontaine - Jacques Beilin

École Nationale des Sciences Géographiques
Pôle Enseignement Géodésie, Mathématiques, Métrologie, Topométrie

9 octobre 2024

Introduction

Objectifs du TD

Le but de ce TD est de calculer la position d'un récepteur GNSS à partir des observations de code C/A et de phase sous Python. Le positionnement pourra être réalisé à partir des mesures de trois constellations GNSS : GPS, Glonass ainsi que Galileo.

Le package python `pygnsstoolbox` comprend différentes fonctions permettant la lecture et l'accès aux données d'observation et de navigation `RINEX`, ainsi que le traitement de ces données et la visualisation des résultats. Les différentes étapes du calcul d'une position pourront alors soit être réalisées dans le cadre du TP, soit s'appuyer sur les fonctions de la bibliothèque.

Prérequis

Des notions de positionnement GNSS sont nécessaires, ainsi qu'une bonne connaissance du langage de programmation scientifique Python/numpy.

Ce TP pourra être réalisé sous Python3, et nécessite l'installation de deux bibliothèques : la bibliothèque `gpsdatetime` pour la gestion des unités de temps ainsi que la bibliothèque `pygnsstoolbox` pour le calcul de positionnement GNSS. L'installation de ces bibliothèques est décrite dans le chapitre suivant.

Documents utiles

Documentation des bibliothèques utilisées

— Documentation de la bibliothèque `pygnsstoobox`, Calcul GNSS sous Python, Manuel utilisateur, ENSG

— Documentation de la bibliothèque `gpsdatetime`, Gestion des dates/heures GPS sous Python, Manuel utilisateur, ENSG

Documents de description du format RINEX

— Document de description du format `RINEX 2.10` : `ftp://igscb.jpl.nasa.gov/pub/data/format/rinex210.txt`

— Document de description du format `RINEX 3.02` : `ftp://igscb.jpl.nasa.gov/pub/data/format/rinex302.pdf`

— Description interactive du format RINEX : `http://gage14.upc.es/gLAB/HTML/LaunchHTML.html`

Documents de spécification de l'interface utilisateur des différents systèmes GNSS

— IS-GPS-200 : document de spécification de l'intreface utilisateur du système GPS, NGA http://www.gps.gov/technical/icwg/IS-GPS-200F.pdf

— Glonass-ICD-5.1 : document de spécification de l'intreface utilisateur du système Glonass, Russian Institute of Space Device Engineering http://www.spacecorp.ru/en/directions/glonass/control_document/

— Galileo OS SIS ICD : document de spécification de l'intreface utilisateur du système Galileo, Galileo Open Service http://ec.europa.eu/enterprise/policies/satnav/galileo/open-service/

Autres documents

— Navipedia : wiki de l'ESA sur les techniques GNSS : http://www.navipedia.net/index.php/Main_Page

Données nécessaires

Un jeu test est disponible dans le répertoire `data`. Il se compose de fichiers `RINEX` d'observations GNSS (extension `.yyo`, où `yy` correspond aux deux dernier chiffres de l'année des observations GNSS), de fichiers `RINEX` de navigation (extension `.yyn` pour un fichier de navigation GPS, `.yyg` pour Glonass et `.yyl` pour Galileo. Extension `.yyp` pour un fichier de navigation multi-constellations), ainsi que de fichiers d'obites précises (extension `.sp3`).

D'autres données peuvent être téléchargées sur les ftp suivant :

— Fichiers d'observation `RINEX` multi-constellation : http://rgp.ign.fr/DONNEES/diffusion/ ou ftp://rgpdata.ign.fr/../pub/data_v3

— Fichiers de navigation `RINEX` multi-constellation ('brdm____.yyp') : ftp://igs.ensg.ign.fr/pub/igs/data/campaign/mgex/daily/rinex3

— Orbites précises : ftp://igs.ensg.ign.fr/pub/igs/products

Déroulement du TD

Le TD est découpé en différentes parties de niveaux différents. Il est possible de le réaliser dans son intégralité ou bien de ne réaliser que certaines parties. Différents 'parcours' possibles sont donnés en annexe.

Déroulement des exercices

Pour chaque exercice, il sera possible :

— d'utiliser directement les fonctions de la bibliothèque
— d'utiliser les squelettes de fonctions développées, qui serviront de point de départ à la programmation de l'exercice
— de réaliser les exercices sans aide, ce qui permettra de prendre plus de libertés par rapport au TD

Prerequis

Installation

Pour installer gpsdatetime, exécuter la commande suivante dans un terminal :

```
sudo pip3 install --break-system-packages gpsdatetime
```

Si vous vous trouvez derrière un proxy, par exemple à l'ENSG :

```
  sudo pip3 --proxy=10.0.4.2:3128 install gpsdatetime
```

Faire de même avec le package gnsstoolbox.

```
sudo pip3 install --break-system-packages gnsstoolbox
```

Pour les installations sous windows avec anaconda, ces commandes doivent être lancées dans le terminal "anaconda prompt".

La bibliothèque gpsdatetime

Le module gpsdatetime est utilisé pour la gestion du temps. La classe *gpsdatetime* contient les attributs suivants :

```
Python 3.5.2 (default, Sep 10 2016, 08:21:44)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import gpsdatetime as gpst
>>> t1 = gpst.gpsdatetime()
>>> print(t1)
Gpstime (version 2016-06-27)
---------------------------------------------------------------
s1970 : 1480065787.511347
YYYY_MM_DD : 2016/11/25
HH:MM:SS : 09:23:07.511347055
GPS week : 1924
Day of week : 5 (FRI)
```

```
Second of week : 465787.511347055
Second of day : 33787.511347055
session : j
Modified Julian Date : 57717.391059
Julian Date : 2457717.891059
YYYY : 2016  DOY : 330
GMST (dec. hour) : 13.702382
GAST (dec. hour) : 13.702256
Eq. of Equinoxes (dec. hour) : -0.000126
----------------------------------------------------------------
```

champs	Description	Unité
s1970	secondes écoulées depuis le 1 janvier 1970 à 0h00Z	seconde
mjd	Modified Julian Date (MJD = JD – 2400000.5)	jours décimaux
jd	Julian Date	jours décimaux
jd50	Julian Date depuis J1950	jours décimaux
wk	Semaine GPS	semaine
wsec	seconde dans la semaine GPS	float [0..604800]
yyyy	année sur 4 chiffres	année I4 [1902..2079]
yy	année sur 2 chiffres	année (I2)
mon	mois de 1 à 12	I2 [0..12]
dd	jour dans le mois	I2
hh	heure	I2 [0..24]
min	minute	I2 [0..60]
sec	seconde	I2 [0..60]
doy	jour dans l'année	I3 [1..366]
wd	jour dans la semaine	I1 [0..6]
dsec	seconde dans la journée	float [0..86400]
dy	année décimale	float [0..366]
GMST	Greenwich Mean Sidereal Time	heures décimales
EQEQ	Equation of Equinoxes	heures décimales
GAST	Greenwich Aparent Sidereal Time	heures décimales

Il est ensuite possible d'accéder à chacune des informations.

```
Python 3.5.2
>>> import gpsdatetime as gpst
>>> t1 = gpst.gpsdatetime()
>>> t1.wsec
465787.511347055
>>> d = t1.doy
>>> print("Day of year : %d" % (d))
Day of year : 330
```

Une explication plus approfondie, ainsi que les syntaxes liées à la structure *gpsdatetime* sont rappelées dans la documentation utilisateur du module.

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP — Université de Technologie Ouverte Pluripartenaire
Partie : 1	Chargement et accès aux données	Difficulté : ⋆

Le chargement et l'accès aux données n'ont pas à être redéveloppés. Différentes fonctions de la bibliothèque `pygpstoolbox` réalisent ces opérations.

Chargement des données

Les fichiers d'observation sont lus avec la fonction `load_rinex_o()`, qui prend en entrée le nom du fichier d'observations `RINEX`.

Les fichiers de navigation sont lus avec la fonction `load_rinex_n()`, qui prend en entrée le nom du fichier de navigation `RINEX`.

Les fichiers contenant les orbites précises sont lus avec la fonction `load_sp3()`, qui prend en entrée le nom du fichier d'orbites précises sp3.

Accès aux données

Les structures précedemment générées n'ont pas à être manipulées directement. Les fonction `get_obs()` et `get_ephemeris()` permettent de récupérer respectivement les observations (pseudo-distance, phase, etc) et éphémérides (éléments képlériens ou position pour intégration de l'orbite) pour un satellite donné et à une date donnée.

La fonction `get_sp3()` permet la récupération d'un arc d'orbite pour un satellite donné, à partir d'une structure $sp3\_data$.

Le contenu des différentes structures générées, ainsi que la syntaxe complète sont précisés dans la documentation utilisateur de la bibliothèque `pygpstoolbox` (fonctions `get`).

À programmer...

Écrire un script `run_calcul`, qui charge les différents fichiers tests :

- fichiers d'observation `RINEX` : `mlvl1500.13o` et `smne1500.13o`
- fichier de navigation `RINEX` : `brdm1500.13p`
- fichiers contenant les orbites précises `sp3` : `igs17424.sp3`, `igl17424.sp3` et

grm17424.sp3 (un fichier par constellation)

Les fichiers sp3 pourront être chargés dans une même structure en utilisant la syntaxe suivante :

```
1
2 mysp3 = orbit()
3 mysp3.load_sp3(nom_fichier1, nom_fichier2, nom_fichier3)
4
5
```

 À tester...

Tester ensuite les fonctions d'accès aux données :

```
1  myrinex = rx.rinex_o()
2  filename = '../../data/mlvl1500.13o'
3  ret = myrinex.load_rinex_o(filename)
4
5  if ret<0:
6      print(ret)
7      return
8
9  t=gpst.gpsdatetime()
10 t.rinex_t('13  5 30  1  0 30.0000000')
11
12 Ep = myrinex.get_epoch_by_mjd(t.mjd)
13
14 for S in Ep.satellites:
15
16     observable = 'C1'
17     S.PR = S.obs.get(observable)
18
19
```

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 2	Introduction au calcul d'orbites	Difficulté : ⋆

Le positionnement GNSS se base sur la mesure du temps de propagation d'une onde électromagnétique entre différents satellites et le récepteur. Il est ainsi nécessaire de connaître la position de chaque satellite à l'instant d'émission du signal. Elles peuvent être calculées à partir d'un fichier de navigation `brdm` ou bien à partir d'un fichier d'orbites précises `sp3`.

> `pygpstoolbox` ...
>
> Ce calcul peut être réalisé par la fonction

```
X, Y, Z, dte = MyOrbit.orb_sat('E', 11, 56442.0)
```

> Remarque ...
>
> Le référentiel dans lequel est exprimée la position de la station est celui dans lequel sont exprimées les positions des satellites. Ainsi, pour une position mesurée uniquement avec des satellites GPS, le système de référence sera le WGS84 si des orbites radiodiffusées ont été utilisées, ou bien l'IGS14 si le calcul a été réalisé avec des orbites précises.

Deux types d'éphémérides sont diffusées :

- les éphémérides des satellites GPS et Galileo sont données sous la forme d'éléments képlériens. Les relations de mécanique céleste permettent de déterminer la position des satellites à un instant donné. Ceci sera réalisé dans le TD 3.

- les éphémérides des satellites Glonass sont diffusées sous forme de positions, vitesses et accélérations. Il est alors nécessaire d'intégrer numériquement les équations différentielles qui décrivent le mouvement des satellites pour connaître leur position à un instant donné. Ceci sera réalisé dans le TD 4.

Afin que le calcul d'orbite puisse être réutilisé lors du calcul de la position d'un récepteur GNSS, il convient de développer une fonction de type :

$$[X, Y, Z] = MyOrbit.pos\_sat(Eph, t)$$

où Eph correspond à un bloc de message de navigation pour un satellite (Eph est fournie par la fonction `get_ephemeris()`).

 À programmer...

Créer la fonction `pos_sat()` prenant en entrée une structure Eph et un temps t donné en Jour Julien Modifié (mjd). Cette fonction sera complétée dans les TD 3 et 4.

ENSG Géomatique ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 3	Calcul de la position d'un satellite GPS ou Galileo à partir d'éléments képlériens	Difficulté : ⋆⋆

La trajectoire d'un satellite est définie par 6 éléments képlériens $(a, e, i, \Omega, \varpi, t_p)$. Sa position sur la trajectoire dans le plan orbital peut être connue à l'aide du rayon r (distance Terre-Satellite) paramétré par l'anomalie vraie v, l'anomalie excentrique E, ou l'anomalie moyenne M.

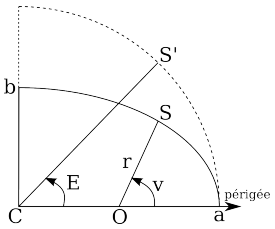


FIG 1 - Paramétrages par v et E de la trajectoire du satellite dans le plan orbital.

Nous utiliserons ici un paramétrage légèrement différent, bien que se basant sur les mêmes éléments de base ; ainsi la position dans le plan orbital sera décrite par l'argument de longitude du satellite $\phi = \varpi + v$ où ϖ désigne l'argument du périgée.

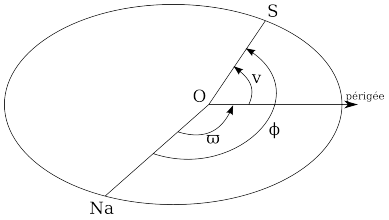


FIG 2 - Paramétrage par ϕ de la trajectoire du satellite dans le plan orbital. Na : Noeud Ascendant.

Le plan orbital sera quant à lui défini par l'inclinaison du plan d'orbite i et la longitude du nœud ascendant Ω.

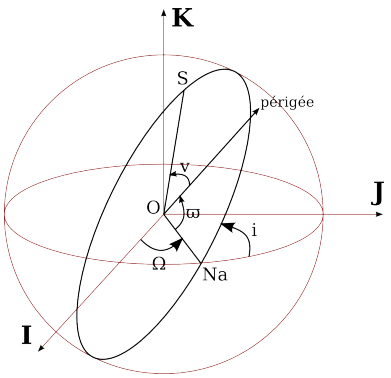


FIG 3 - Éléments képlériens d'un satellite S en orbite autour de la Terre.

Les différents éléments képlériens sont donnés dans la structure Eph (exemple d'une éphéméride d'un satellite GPS) :

```
Attributs d'un objet Eph
Eph =


    PRN =  14
    mjd =  56442
    TOC =  56442
    alpha0 =  2.20693647861500e-04
    alpha1 = -2.27373675443200e-13
    alpha2 = 0
    const = G
    IODE =  80
    crs =  19.0937500000000
    delta_n =  3.88909056766100e-09
    M0 =  2.69875463238600
    cuc =  1.01700425148000e-06
    e =  0.00693027686793400
    cus =  1.04121863842000e-05
    sqrt_a =  5153.75222969100
    TOE =  345600
    cic =  9.12696123123200e-08
    OMEGA =  0.698653185092300
    cis = -7.82310962677000e-08
    i0 =  0.974451555395800
    crc =  184
    omega = -2.03491600246500
    OMEGA_DOT = -7.63674667258100e-09
    IDOT = -3.27513642257800e-10
    code_L2 = 0
    gps_wk =  1742  ( = gal_wk dans le cas de Galileo)
    L2_P = 0
    sv_acc =  2
    sv_health = 0
    TGD = -8.84756445884700e-09
```

```
IODC =  80
trans_time =  342126
```

 pygnsstoolbox ...

Ce calcul peut être réalisé par la fonction suivante

```
1 X,Y,Z,dte = Orb.calcSatCoord(constellation,prn,t.mjd)
```

 À programmer...

Le calcul d'orbites à partir d'éléments képlériens est à développer dans la fonction `pos_sat_brdc()`.

Étape 1 : Passage des éléments d'orbite à la position sur l'orbite

Les éléments donnés dans le fichier `RINEX` de navigation sont ceux valables à une date donnée appelée "date de référence des éphémérides" TOE (Time of Ephemeris).

 A propos de TOE, TOC, mjd...

Dans la toolbox, les champs mjd et TOC renvoient la même valeur, le Time-of-Clock exprimé en jour julien modifié. Le TOE renvoie en pratique sur le même instant, mais exprimé en seconde dans la semaine. Pour obtenir une date définie de maniière complète, il faut donc associer ce champ au numéro de semaine GPS.

 À programmer...

Calculer le temps orbital Δt par rapport au temps usuel t à l'aide de la relation

$$\Delta t = t - TOE$$

On prendra soin de travailler dans la même échelle de temps pour les deux grandeurs.

Calcul du moyen mouvement

La trajectoire du satellite n'étant pas tout à fait elliptique, la vitesse angulaire de mouvement moyen doit être corrigée par rapport à la valeur théorique.

À programmer...

Calculer le moyen mouvement n

$$n = n_0 + \Delta n \text{ avec } n_0 = \sqrt{\frac{\mu}{a^3}}$$

où μ est la constante gravitationnelle de la Terre et Δn une correction issue du fichier RINEX.

$$\mu = GM = 3.986005.10^{14}\ \mathrm{m}^3.\mathrm{s}^{-2}$$

Calcul de l'anomalie moyenne à la date t

À programmer...

Calculer l'anomalie moyenne M à l'instant t_0

Elle est donnée par

$$M = M_0 + n \times \Delta t$$

où M_0 est l'anomalie moyenne au temps de référence des éphémérides fourni par le RINEX.

Résolution de l'équation de Kepler

L'équation de Kepler s'écrit

$$M = E - e \times \sin E$$

où e est l'excentricité de l'orbite du satellite fournie dans le fichier RINEX et E l'anomalie excentrique, à calculer.

À programmer...

Cette équation peut être résolue par itération.
En posant $E_0 = M$, on calcule la nouvelle estimation de E par

$$E_{k+1} = M + e \times \sin E_k$$

Arrêtez le calcul dès que vous obtenez une précision centimétrique.

Calcul de l'anomalie vraie du satellite

À programmer...

Calculer l'anomalie vraie v à partir de l'anomalie excentrique. Elle est déduite de la relation

$$\tan\frac{v}{2} = \sqrt{\frac{1+e}{1-e}}\tan\frac{E}{2}$$

Calcul du rayon Terre-Satellite r

On rappelle que la relation entre le rayon r et l'anomalie excentrique E est donnée par

$$r = a\,(1 - e\cos E)$$

À programmer...

Calculer le rayon r.

Calcul des coordonnées dans le plan orbital

Les coordonnées orbitales sont calculées à partir des coordonnées polaires (r, ϕ). Cependant la trajectoire étant **perturbée**, elle n'est pas parfaitement elliptique.

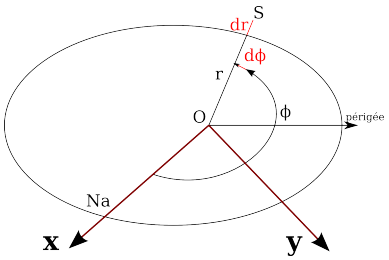


FIG 4 - Trajectoire perturbée du satellite dans le repère orbital.

À programmer...

Les coordonnées polaires sont données en première approximation par $\phi = \varpi + v$ et $r = a\,(1 - e\cos E)$. ϖ est représenté par $omega$ dans le message de navigation.

Les termes ϕ et r doivent cependant être corrigés de la façon suivante :

$$\delta\phi = C_{us}\sin 2\phi + C_{uc}\cos 2\phi$$

$$\delta r = C_{rs} \sin 2\phi + C_{rc} \cos 2\phi$$

où $C_{us}, C_{uc}, C_{rs}, C_{rc}$ sont des coefficients issus du fichier `RINEX` de navigation.

Les coordonnées dans le plan orbital où l'axe OX est dirigé vers le nœud ascendant, s'expriment par :

$$x = (r + \delta r) \cos(\phi + \delta\phi)$$

$$y = (r + \delta r) \sin(\phi + \delta\phi)$$

Étape 2 : Position du plan orbital dans l'espace

Le plan orbital subit lui aussi des perturbations diverses. Il en résulte que d'une part, il se déplace au cours du temps et que, d'autre part, ce n'est pas tout à fait un plan.

Ω_0 est représenté par $OMEGA$ dans le message de navigation.

On la modélise par la vitesse linéaire $\dot{i}$ ($IDOT$) de l'inclinaison du plan orbital et par la vitesse $\dot{\Omega}$ ($OMEGA\_DOT$) du nœud ascendant.

La non-planéité du plan est modélisée par des harmoniques sphériques sur i en fonction de l'argument de longitude du satellite.

À programmer...

$$i = i_0 + \dot{i} \times \Delta t + \delta i$$

$$\Omega = \Omega_0 + \dot{\Omega} \times \Delta t$$

$$\text{avec } \delta i = C_{is} \sin 2\phi + C_{ic} \cos 2\phi$$

où C_{is}, C_{ic} sont des coefficients issus du fichier `RINEX` de navigation.

Passage aux coordonnées cartésiennes géocentriques ECI (Earth-Centered-Inertial)

On a donc maintenant accès à la fois à la position du satellite dans le plan orbital et à la position de ce plan dans l'espace. Le passage des coordonnées exprimées dans le repère orbital aux coordonnées exprimées dans le repère équatorial céleste (ECI) se fait alors par deux rotations : l'une d'angle i autour de l'axe des nœuds, l'autre d'angle Ω autour de l'axe de rotation de la Terre.

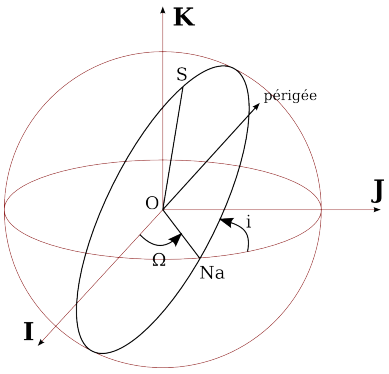


FIG 5 - Position du plan orbital dans l'espace.

À programmer...

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{ECI} = \begin{bmatrix} \cos\Omega & -\sin\Omega & 0 \\ \sin\Omega & \cos\Omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos i & -\sin i \\ 0 & \sin i & \cos i \end{bmatrix} \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}_{orb}$$

Passage aux coordonnées cartésiennes ECEF (Earth-Fixed-Earth-Centred)

Il reste alors à passer du repère équatorial céleste au système WGS84 (pour GPS) ou GTRF (pour Galileo) de la date t.

Les systèmes WGS84 et GTRF sont aussi équatoriaux mais leur axe-origine est contenu dans le plan du méridien de Greenwich. Par conséquent, il faut et il suffit d'effectuer une rotation autour de l'axe de rotation terrestre d'une valeur corrigeant de la rotation de la Terre à la date t.

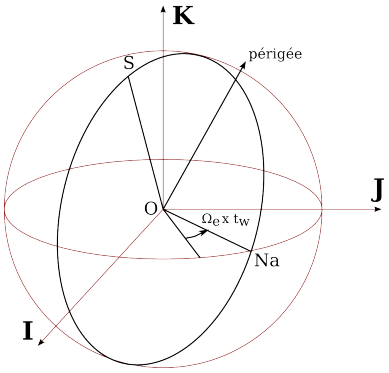


FIG 6 - Transformation des coordonnées ECI en coordonnées ECEF.

À programmer...

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{ECEF} = \begin{bmatrix} \cos(\dot{\Omega}_e \times t) & -\sin(\dot{\Omega}_e \times t) & 0 \\ \sin(\dot{\Omega}_e \times t) & \cos(\dot{\Omega}_e \times t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{ECI}$$

où $\dot{\Omega}_e = -7.2921151467.10^{-5}\ \mathrm{rad.s^{-1}}$ est la vitesse angulaire de rotation de la Terre. Le temps t utilisé ici pour calculer la matrice rotation est le nombre de secondes dans la semaine GPS/Galileo (durée entre le dimanche précédent à 0h et l'instant pour lequel on cherche la position).

Attention : Si l'éphéméride date de la semaine GPS précédente à celle de l'observation (par exemple, observation le dimanche à 0h12, et éphéméride du samedi à 23h59), ne pas oublier d'ajouter 86400×7 secondes au temps t.

Si au contraire, le mjd et la semaine GPS de l'observation précèdent le mjd et la semaine GPS de l'éphéméride utilisée, soustraire 86400×7 secondes au temps t.

Vous pouvez maintenant calculer la position de chaque satellite GPS et Galileo dans le référentiel associé à la constellation (WGS84 pour GPS et GTRF pour Galileo).

À tester...

Dans le script `Tp3OrbitsNav.py`, compléter la fonction `pos_sat_brdc`. Récupérer l'éphéméride du satellite G14, et calculez la position de ce satellite pour le jour 240/2021 à 1h30min35s.

```
""" Gestions des ephemerides radiodiffusees (fichiers "n")
    et precises sp3 """
import gnsstoolbox.orbits as orbits

Orb = orbits.orbit()

""" lecture du fichier BRDC """
dir_orb = '../data'
Orb.loadRinexN(os.path.join(dir_orb, '
    MLVL00FRA_R_20212400000_01D_GN.rnx'))

""" Definition de l'instant pour lequel on cherche une
    position """
t = gpst.gpsdatetime(yyyy=2021,doy=240,dsec=5435)
print(t)
```

```
13
"""  SATELLITE  """
15 constellation = 'G'
prn = 14
17
try:
19 Eph = Orb.getEphemeris(constellation,prn,t.mjd)
print(Eph)
21 print("TOC : ",Eph.tgps.st_iso_epoch())

23 except:
print("Unable to find satellite !!! ")
25
print("\nCalcul avec la fonction integee a la toolbox")
27 X,Y,Z,dte = Orb.calcSatCoord(constellation,prn,t.mjd)
print("Solution pygnssToolbox\nX = %13.3f m\nY = %13.3f m\nZ
    = %13.3f m\ndte = %.9f s" % (X,Y,Z,dte))
29
""" impression des elements de debug """
31 #     print(Orb.debug.__dict__)

33 print("\nCalcul avec la fonction developpee lors du TP")
X,Y,Z,dte = tp3_pos_sat_brdc(Orb, constellation,prn,t.mjd)
35 print("Solution TP\nX = %13.3f m\nY = %13.3f m\nZ = %13.3f m
   \ndte = %.9f s" % (X,Y,Z,dte))
```

Résultat :

```
X = -12612103.330 m
Y =  20676370.840 m
Z = -10845012.324 m
dte = 0.000014084 s
```

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 4	Calcul de la position d'un satellite Glonass par intégration numérique	Difficulté : ⋆⋆⋆

A mettre à jour / rinex3

Les éphémérides Glonass sont données sous la forme de coordonnées (x, y et z) et vitesses (v_x, v_y et v_z) des satellites à un instant donné (TOE), ainsi que du vecteur accélération lié à la perturbation gravitationnelle luni-solaire (X", Y" et Z"). La position des satellites sera donc calculée à l'instant t en intégrant numériquement les équations différentielles qui décrivent le mouvement des satellites. Les coordonnées et vitesses diffusées dans le message de navigation forment les conditions initiales.

Les différentes positions, vitesses et accélérations sont donnés dans la structure Eph (exemple d'une éphéméride d'un satellite Glonass) :

```
Eph =

  scalar structure containing the fields:

    PRN =  14
    mjd =  56442.3020833335
    TOC =  56442.3020833335
    const = R
    SV_clock_offset = -3.77744436264000e-06
    SV_relat_freq_offset = -9.09494701772900e-13
    Message_frame_time =  370800
    X =  24284381.3476600
    X_dot =  925.148010253900
    MS_X_acc =  1.86264514923100e-06
    sv_health = 0
    Y = -4020694.82421900
    Y_dot = -276.086807251000
    MS_Y_acc = -2.79396772384600e-06
    freq_num = -7
    Z = -6860929.68750000
    Z_dot =  3442.86537170400
    MS_Z_acc = 0
    age_op_inf = 0
```

pygpstoolbox ...

Ce calcul peut être réalisé par la fonction

```
[mjd,X,Y,Z,VX,VY,VZ,dte,debug] = orb_from_RK4(Eph,t)
```

À programmer...

Le calcul d'orbites par intégration numérique est à développer dans la fonction `pos_sat()`.

Étape 1 : Transformation des coordonnées dans un référentiel inertiel (ECI)

De même que pour le calcul d'orbites à partir d'éléments képlériens, les éléments donnés dans le fichier `RINEX` de navigation sont ceux valables à une date donnée appelée "date de référence des éphémérides" TOE (Time of Ephemeris).

À programmer...

Calculer le temps orbital t_o par rapport au temps usuel t à l'aide de la relation

$$t_o = t - TOE$$

On prendra soin de travailler dans la même échelle de temps pour les deux grandeurs.

Les conditions initiales (x, y, z, v_x, v_y et v_z) sont données dans le référentiel ECEF (Earth-Centred-Earth-Fixed) PZ-90. Elle doivent donc être transformées dans un système de coordonnées inertiel pour pouvoir être utilisées dans l'intégration. On effectue ainsi une rotation des coordonnées d'un angle :

$$\theta_{Ge} = \theta_{G0} + \Omega_e \times t_0$$

où θ_{G0} est le temps sidéral à l'époque t_0, θ_{Ge} correspond au temps sidéral à Greenwich à minuit et Ω_e est la vitesse angulaire de rotation de la Terre.

Les composantes de la vitesse sont calculées en appliquant la formule de dérivation vectorielle :

$$\left(\frac{\mathrm{d}\vec{x}}{\mathrm{d}t}\right)_{ECI} = \left(\frac{\mathrm{d}\vec{x}}{\mathrm{d}t}\right)_{ECEF} + \vec{\Omega}_e \wedge \vec{x}_{ECI}$$

avec $\vec{\Omega}_e = \begin{pmatrix} 0 \\ 0 \\ \Omega_e \end{pmatrix}$.

À programmer...

Transformer les coordonnées initiales dans un repère inertiel :
Position :

$$\begin{cases} x_a(t_0) &= x(t_0)\cos(\theta_{Ge}) - y(t_0)\sin(\theta_{Ge}) \\ y_a(t_0) &= x(t_0)\sin(\theta_{Ge}) + y(t_0)\cos(\theta_{Ge}) \\ z_a(t_0) &= z(t_0) \end{cases}$$

Vitesse :

$$\begin{cases} v_{xa}(t_0) &= v_x(t_0)\cos(\theta_{Ge}) - v_y(t_0)\sin(\theta_{Ge}) - \Omega_e y_a(t_0) \\ v_{ya}(t_0) &= v_x(t_0)\sin(\theta_{Ge}) + v_y(t_0)\cos(\theta_{Ge}) + \Omega_e x_a(t_0) \\ v_{za}(t_0) &= v_z(t_0) \end{cases}$$

On rappelle que $\Omega_e = 7.2921151467.10^{-5}$ rad. s^{-1}. Le temps sidéral θ_{G0} correspond au champs $GAST$ (donné en heure) d'une structure mjd (cf. doc. `gpsdatetime`).

De même, le vecteur accélération lié à la perturbation gravitationnelle luni-solaire (X", Y" et Z") diffusé par le message de navigation est une projection du vecteur accélération luni-solaire sur les axes du référentiel ECEF. Il doit aussi être transformé dans un référentiel inertiel, ce qui est réalisé en effectuant une rotation d'axe θ_{Ge}.

À programmer...
Transformer le vecteur accélération luni-solaire dans un repère inertiel :

$$\begin{cases} J_{xa} &= X"(t_0)\cos(\theta_{Ge}) - Y"(t_0)\sin(\theta_{Ge}) \\ J_{ya} &= X"(t_0)\sin(\theta_{Ge}) + Y"(t_0)\cos(\theta_{Ge}) \\ J_{za} &= Z"(t_0) \end{cases}$$

X", Y" et Z" correspondent aux champs MS_X_acc, MS_Y_acc et MS_Z_acc de la structure Eph.

Étape 2 : Intégration numérique des équations différentielles décrivant le mouvement des satellites

Le système à intégrer décrit un mouvement képlérien (donc à deux corps : Terre-Satellite) auquel ont été ajouté l'influence du C_{20} ($= -J_2$, influence de l'applatissement de la Terre aux pôles) et de l'attraction luni-solaire.

$$\begin{cases} \frac{\mathrm{d}x_a}{\mathrm{d}t} &= v_{xa}(t) \\ \frac{\mathrm{d}y_a}{\mathrm{d}t} &= v_{ya}(t) \\ \frac{\mathrm{d}z_a}{\mathrm{d}t} &= v_{za}(t) \\ \frac{\mathrm{d}v_{xa}}{\mathrm{d}t} &= -\bar{\mu}\bar{x}_a + \frac{3}{2}C_{20}\bar{\mu}\bar{x}_a\rho^2(1-5\bar{z}_a^2) + J_{xa} \\ \frac{\mathrm{d}v_{ya}}{\mathrm{d}t} &= -\bar{\mu}\bar{y}_a + \frac{3}{2}C_{20}\bar{\mu}\bar{y}_a\rho^2(1-5\bar{z}_a^2) + J_{ya} \\ \frac{\mathrm{d}v_{za}}{\mathrm{d}t} &= -\bar{\mu}\bar{z}_a + \frac{3}{2}C_{20}\bar{\mu}\bar{z}_a\rho^2(3-5\bar{z}_a^2) + J_{za} \end{cases}$$

avec :

$$\bar{\mu} = \frac{\mu}{r^2}, \bar{x}_a = \frac{x_a}{r}, \bar{y}_a = \frac{y_a}{r}, \bar{z}_a = \frac{z_a}{r}, \bar{\rho} = \frac{a_E}{r}, r = \sqrt{x_a^2 + y_a^2 + z_a^2}$$

où

- $a_E = 6\,378{,}136\,\mathrm{km}$: Demi grand-axe de la Terre (PZ-90)
- $\mu = 398\,600{,}44\,\mathrm{km}^3.\mathrm{s}^{-2}$: Constante gravitationnelle (PZ-90)
- $C_{20} = -1\,082{,}63 \cdot 10^{-6}$: Second coefficient zonal de la décomposition en harmoniques sphériques du potentiel gravitationnel

Ce système d'équations ne pouvant être résolu analytiquement, nous utiliserons l'algorithme **Runge-Kutta 4** pour déterminer la valeur des différentes inconnues à un instant t par intégration numérique.

Intégration numérique par la méthode Runge-Kutta 4

On cherche à résoudre un système d'équations différentielles :

$$\begin{cases} \frac{\mathrm{d}y_1}{\mathrm{d}t} &= f_1(t, y_1, \ldots, y_n) \\ &\vdots \\ \frac{\mathrm{d}y_n}{\mathrm{d}t} &= f_n(t, y_1, \ldots, y_n) \end{cases}$$

Ce qui est équivalent à :

$$\mathbf{Y}'(t) = \mathbf{F}(t, \mathbf{Y}(t))$$

$\mathbf{Y}(t_0)$ est connu, et forme la condition initiale du problème. On recherche $\mathbf{Y}(t_f)$ à un instant final. L'intégration est réalisée de manière discrète : le vecteur $\mathbf{Y}$ sera calculé à chaque instant t_k, où $t_{k+1} - t_k = h$ est le pas d'intégration.

La méthode **Runge-Kutta 4** permet ainsi de calculer une solution $\mathbf{Y}(t_{k+1})$ à partir de la solution $\mathbf{Y}(t_k)$.

Elle se base sur l'algorithme suivant :

$$\begin{aligned}
\mathbf{K}_1 &= \mathbf{F}(t_k, \mathbf{Y}_k) \\
\mathbf{K}_2 &= \mathbf{F}(t_k + h/2, \mathbf{Y}_k + h\mathbf{K}_1/2) \\
\mathbf{K}_3 &= \mathbf{F}(t_k + h/2, \mathbf{Y}_k + h\mathbf{K}_2/2) \\
\mathbf{K}_4 &= \mathbf{F}(t_k + h, \mathbf{Y}_k + h\mathbf{K}_3) \\
\mathbf{Y}_{k+1} &= \mathbf{Y}_k + h/6(\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4)
\end{aligned}$$

À programmer...

Implémenter l'algorithme Runge-Kutta 4 :

Les valeurs initiales sont données dans la structure Eph. On appliquera l'algorithme jusqu'à atteindre la date t, en partant de la date TOE de l'éphéméride.

On pourra prendre un pas d'intégration de 150 secondes. Le dernier pas d'intégration sera modifié de manière à atteindre la date t.

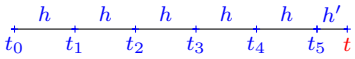


FIG 7 - Pas d'intégration h entre t_0 et t_5, et h' entre t_5 et t.

Étape 3 : Transformation des coordonnées dans le référentiel PZ-90

Une fois les coordonnées et vitesses (x, y, z, v_x, v_y et v_z) calculées à la date t, il reste à les transformer dans le référentiel PZ-90. On applique ainsi la transfomation développée dans l'étape 1, tout en remplaçant θ_{Ge} par $-\theta_{Ge}$.

À programmer...

Transformer les coordonnées et vitesses dans le référentiel PZ-90.

On pourra éventuellement les transformer en WGS84 par la formule suivante :

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{WGS84} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{PZ-90} + \begin{bmatrix} -0.36m \\ 0.08m \\ 0.18m \end{bmatrix}$$

À tester...

Dans le script `Tp4OrbitsRk4.py`, compléter la fonction `pos_sat_rk4`. Récupérer l'éphéméride du satellite R01, et calculez la position de ce satellite pour le jour 285/2018 à 1h30 :

```
""" Gestions des ephemerides radiodiffusees (fichiers "n")
    et precises sp3 """
import gnsstoolbox.orbits as orbits

Orb = orbits.orbit()

""" lecture du fichier BRDC """
dir_orb = '../data'
Orb.loadRinexN(os.path.join(dir_orb, '
    MLVL00FRA_R_20212400000_01D_RN.rnx'))

""" Definition de l'instant pour lequel on cherche une
    position """
t = gpst.gpsdatetime(yyyy=2021,doy=240,dsec=5400)

""" Ecart temps Glonass/GPS """
t -= 18
print(t)

""" SATELLITE """
constellation = 'R'
prn = 1

try:
Eph = Orb.getEphemeris(constellation,prn,t.mjd)
print(Eph)
print("TOC : ",Eph.tgps.st_iso_epoch())
except:
print("Unable to find satellite !!! ")

XSP3 =[  15877666.256,  -3409654.358, -19677679.894,
    84.494967]

""" Calcul avec la fonction integree a la toolbox """
X,Y,Z, VX,VY,VZ,dte, deb = Orb.calcSatCoordGlonassNav(
    constellation,prn,t.mjd)
print("\nSolution pygnssToolbox\nX = %13.3f \nY = %13.3f \nZ
    = %13.3f \ndte = %.9f us" % (X,Y,Z,dte))

#    print(X - XSP3[0], Y-XSP3[1], Z-XSP3[2])
```

```
35
""" Calcul avec la fonction developpee lors du TP """
37 X,Y,Z,dte = pos_sat_rk4(Orb, constellation, prn, t.mjd)
print("\nSolution TP\nX = %13.3f \nY = %13.3f \nZ = %13.3f \
    ndte = %.9f us" % (X,Y,Z,dte))
39
```

```
Résultat :
X = -11817486.175 m
Y = 21992000.359 m
Z = -5195631.685 m
```

Pour aller plus loin...

La précision des positions et vitesses intégrées numériquement dépend à la fois de l'intervalle entre la diffusion de deux éphémérides successives (généralement 30 minutes, ce qui implique qu'une éphéméride est disponible à moins de 15 minutes de la date t), ainsi que du pas d'intégration. Un tableau donnant une indication sur la précision à attendre est donnée en Annexe 3 de l'ICD Glonass.

Faire varier le pas d'intégration et comparer les orbites obtenues entre elles.

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 5	Calcul de la position d'un satellite GNSS à partir des éphémérides précises	Difficulté : ⋆⋆

Le but est de déterminer la position d'un satellite GNSS à une époque t à partir des positions données dans un fichier d'éphémérides précises de l'IGS (format sp3). Les éphémérides précises de l'IGS sont données dans le référentiel (Earth Centered - Earth Fixed) IGS14 sous forme de 3 coordonnées X,Y et Z (km) et un décalage d'horloge (μs) toutes les 5 minutes.

Pour obtenir les éphémérides à une époque, on utilise un polynôme de Lagrange.

$$y(t) = \sum_{j=0}^{m} L_j(t).y(t_j)$$

où

$$L_j(t) = \prod_{k=0}^{m} \frac{(t - t_k)}{(t_j - t_k)}, k \neq j$$

où m est l'ordre du polynôme, $y(t_j)$ est la valeur à l'instant t_j, $L_j(t)$ est appelée fonction de base d'ordre m et t est l'instant où la donnée est interpolée.

m est généralement un nombre impair.

Le polynôme de Lagrange est utilisé sur chacune des coordonnées du satellite ainsi que sur l'erreur d'horloge.

`pygnsstoolbox` ...

Ce calcul peut être réalisé par la fonction

```
(X, Y, Z, dte) = mysp3.calcSatCoordSp3(constellation,prn,mjd
    ,ordre)
```

À programmer...

1. Ecrire la fonction d'interpolation

$$[X, Y, Z, dte] = pos\_sat\_sp3(sp3\_data, mjd, const, PRN, ordre)$$

ayant pour paramètres :

— les positions du satellite obtenues en sortie de la fonction `load_sp3()` (structure $sp3\_data$)

— le type de constellation

— le PRN du satellite

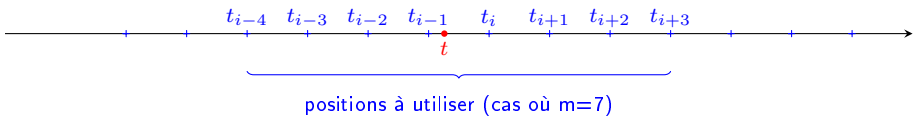
— la date d'interpolation en Jour Julien Modifié

— l'ordre du polynôme.

Elle fournit en sortie les coordonnées (X, Y, Z) et l'erreur d'horloge du satellite dte.

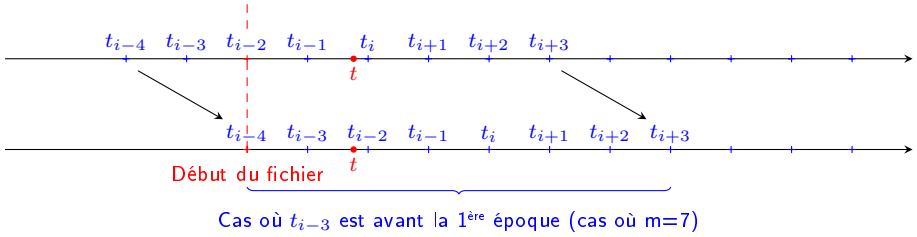
Quelques étapes à respecter pour faciliter la programmation...

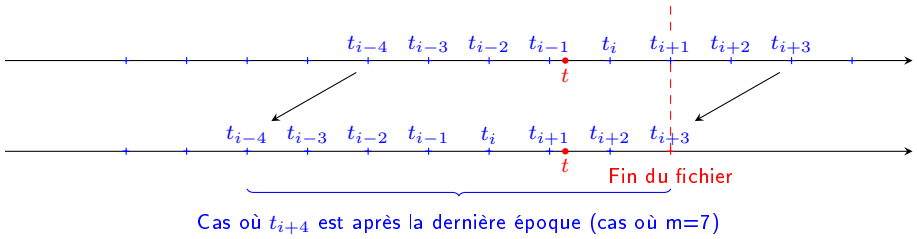
— La première étape consiste à extraire les m+1 époques nécessaires à l'interpolation. Dans le cas général, on cherche l'indice i de l'époque t_i du fichier qui suit juste l'instant t_0 pour lequel on cherche la position du satellite. A partir de cette date, on extrait les époques entre $i - \frac{m+1}{2}$ et $i + \frac{m+1}{2} - 1$. On obtient donc les $m + 1$ époques nécessaires. On pourra s'aider de la fonction `get_sp3()` qui renvoie l'arc d'orbite pour un satellite donné.



— A partir de ce tenseur, on peut appliquer directement l'algorithme.

— Il est par la suite possible de gérer les effets de bords. En effet, si on souhaite interpoler une position proche du début ou de la fin du fichier, on décale l'intervalle pour ne sélectionner que des époques présentes dans le fichier.





2. A partir de la fonction précédente, écrire une fonction qui renvoie le vecteur vitesse (V_X, V_Y, V_Z) du satellite au même instant.

À tester...

Dans le script Tp5OrbitsSp3.py, compléter la fonction pos_sat_sp3. Récupérer l'arc d'orbite du satellite G14, et calculez la position de ce satellite pour le jour 240/2021 à 1h30m35s, avec un polynôme de Lagrange d'ordre 9 :

```
""" Gestions des ephemerides radiodiffusees (fichiers "n")
    et precises sp3 """
mysp3 = orbits.orbit()

dir_orb = '../data'
mysp3.loadSp3(os.path.join(dir_orb,'
    GFZ0OPSULT_20212400600_02D_05M_ORB.SP3'))

""" Definition de l'instant pour lequel on cherche une
    position """
t = gpst.gpsdatetime(yyyy=2021,doy=240,dsec=5435.000)
#     print(t)

""" SATELLITE """
constellation = 'G'
prn = 14

""" Ordre du polynome """
ordre = 9

""" Calcul avec la fonction integree a la toolbox """
(X_sp3,Y_sp3,Z_sp3,dte_sp3)  = mysp3.calcSatCoordSp3(
    constellation,prn,t.mjd,ordre)
print("\nSolution pygnssToolbox\nX = %13.3f m\nY = %13.3f m\
```

```
    nZ = %13.3f m\ndte = %.9f s" % (X_sp3,Y_sp3,Z_sp3,dte_sp3
    ))

(X1,Y1,Z1,dte1) = mysp3.calcSatCoordSp3(constellation,prn,t.
    mjd-0.001/86400,ordre)
(X2,Y2,Z2,dte2) = mysp3.calcSatCoordSp3(constellation,prn,t.
    mjd+0.001/86400,ordre)
V = np.array([[X2-X1],[Y2-Y1],[Z2-Z1]]) / 0.002
print("V = %.3f m/s" % np.linalg.norm(V))

""" Fonction developpee lors du TP """
X,Y,Z,dte = pos_sat_sp3(mysp3, constellation, prn, t.mjd,
    ordre)
print("\nFonction developpee lors du TP\nX = %13.3f m\nY =
    %13.3f m\nZ = %13.3f m\ndte = %.9f s" % (X,Y,Z,dte))
```

Résultat :

```
X = -12612103.423 m
Y =  20676372.825 m
Z = -10845011.399 m
dte = 0.000014083 s
```

Remarque ...

Les orbites sp3 donnent la position du centre de masse des satellites, et non celle du centre de phase comme les orbites radiodiffusées. Il faudrait donc prendre en compte le vecteur entre centre de masse et centre de phase du satellite, donné dans le fichier de calibration `igs14.atx`. Ceci ne sera pas réalisé dans le TD.

ENSG Géomatique ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 6	Visualisation et comparaison des orbites	Difficulté : ⋆

Visualisation des orbites en 3D

Visualiser en 3D l'orbite calculée de satellites GPS ou Galileo sur une période de 24 heures d'abord dans le référentiel celeste (ECI) puis dans le référentiel WGS84 (ECEF). Pour ce faire, on s'appuiera sur le fait que l'algorithme de calcul de la position à partir du message de navigation passe d'abord par le calcul de la position dans un référentiel inertiel avant d'ajouter l'effet de la rotation de la terre.

À programmer...

1. Visualiser en 3D dans le référentiel ECI l'orbite des satellites GPS 1, 3, 14, 7, 12 et 8 (si présents dans les messages de navigation). Ces satellites sont disposés sur chacun des 6 plans orbitaux de la constellation GPS.
2. Représenter les mêmes satellites dans le référentiel ECEF.
3. Visualiser en 3D dans le référentiel ECI l'orbite des satellites Galileo 1, 11 et 5. Ces satellites sont disposés sur chacun des 3 plans orbitaux de la constellation Galileo.
4. Représenter les mêmes satellites dans le référentiel ECEF.

Comparaison entre orbites radiodiffusées et orbites précises

À programmer...
Comparer les coordonnées du satellite G14 obtenues à partir du fichier rinex de navigation (TP3) à celles du fichier sp3. La comparaison sera effectuée au pas de 5 minutes. Il n'est donc pas utile d'interpoler dans le fichier sp3.

Comparaison entre des orbites interpolées par le polynôme de Lagrange à différents ordres

Faire varier l'ordre du polynôme de Lagrange et comparer les orbites obtenues entre elles.

À programmer...
On pourra comparer les ordres 3, 5, 7, 11 et 13 à l'ordre 9. Observer le comportement des orbites calculées au début et à la fin du fichier sp3.

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP — Université de Technologie Ouverte Pluripartenaire
Partie : 7	Écriture d'une fonction de trilatération : I - GPS seul	Difficulté : ⋆⋆

L'équation d'observation de la mesure de pseudo-distance est la suivante :

$$PR = \rho + c \times dt_r - c \times dt_e - c \times dt_{relat} + d_{tropo} + d_{iono}$$

où

- PR : pseudo-distance mesurée (code C/A ou code P)
- $\rho = \sqrt{(X_{rec} - X_{sat})^2 + (Y_{rec} - Y_{sat})^2 + (Z_{rec} - Z_{sat})^2}$: distance géométrique entre le récepteur et un satellite
- dt_r : erreur d'horloge du récepteur
- dt_e : erreur d'horloge du satellite
- dt_{relat} : correction relativiste
- d_{tropo} : retard troposphérique
- d_{iono} : retard ionosphérique
- c : célérité de la lumière dans le vide, $c = 299792458.0\,\mathrm{m.s^{-1}}$

Dans cette équation :

- **Est mesurée** : la pseudo-distance PR
- **Sont connus ou modélisés** : la position du satellite $(X_{sat}, Y_{sat}, Z_{sat})$, l'erreur d'horloge satellite dt_e, la correction relativiste dt_{relat}, ainsi que les retards troposphérique et ionosphérique d_{tropo} et d_{iono}.
- **Sont inconnus et doivent être estimés** : la position du récepteur $(X_{rec}, Y_{rec}, Z_{rec})$, ainsi que l'erreur d'horloge récepteur dt_r.

On peut alors corriger PR d'une partie des erreurs qui sont modélisées. On pose alors

$$PR_{corr} = PR + c \times dt_e + c \times dt_{relat} - d_{tropo} - d_{iono}$$

Ce qui donne l'équation :

$$PR_{corr} = \rho + c \times dt_r$$

$$\Leftrightarrow PR_{corr} = \sqrt{(X_{rec} - X_{sat})^2 + (Y_{rec} - Y_{sat})^2 + (Z_{rec} - Z_{sat})^2} + c \times dt_r$$

Cette équation peut être résolue par moindres carrés, si on dispose d'au minimum 4 observations (ce qui est égal au nombre d'inconnues à estimer).

À programmer...

Il s'agit donc dans un premier temps d'écrire une fonction de calcul de multilatération spatiale par moindres carrés, la fonction :

```
dir_data = '../data'
PosSat, Dobs, X0 = loadJson(os.path.join(dir_data, "data_tp7
    .json"))

X,Y,Z,cdtr,sigma0_2,V,SigmaX = estimation(PosSat,Dobs,X0)
print("\nSolution TP\nX = %13.3f m\nY = %13.3f m\nZ = %13.3f
    m\nc*dtr = %.9f m" % (X,Y,Z,cdtr))
print("sigma0_2 = %.3f" % (sigma0_2))
print("V = ",V,"\nSigmaX = ",SigmaX)
```

En entrée de la fonction :

- Coordonnées des satellites sous forme d'une matrice (j x 3) : $(X_{sat}, Y_{sat}, Z_{sat})^j$
- Pseudo-distances observées et corrigées sous forme d'un vecteur (j x 1) : PR^j_{corr}
- Coordonnées approchées du récepteur et cdtr approché, par défaut : $[0; 0; 0; 0]$

En sortie de la fonction :

- Position du récepteur (X, Y, Z)
- Erreur d'horloge récepteur, en distace : $cdtr$
- Vecteur V des résidus
- Estimateur du facteur unitaire de variance $\hat{\sigma}_0^2 = \frac{V^t P V}{n-p}$
- Matrice de variance-covariance sur les paramètres $\Sigma_{\hat{X}}$

L'équation d'observation à linéariser est :

$$PR^j_{corr} = \sqrt{(X_{rec} - X^j_{sat})^2 + (Y_{rec} - Y^j_{sat})^2 + (Z_{rec} - Z^j_{sat})^2} + c \times dt_r$$

où PR^j_{corr} est la pseudo-distance observée et corrigée entre le récepteur et le satellite j, et $c \times dt_r$ l'erreur systématique effectuée sur toutes les distances due à l'erreur d'horloge récepteur.

Les inconnues à estimer sont les coordonnées du récepteur $(X_{rec}, Y_{rec}, Z_{rec})$, ainsi que l'erreur systématique effectuée sur toutes les distances due à l'erreur d'horloge récepteur $c \times dt_r$.

Le module doit pouvoir fonctionner avec un nombre variable de satellites (minimum 5, afin de pouvoir calculer $\hat{\sigma}_0^2$).

Données test...

Afin de tester la fonction, un jeu de données test a été produit. On détermine la position du récepteur par multilatération à partir de 10 satellites. Les données sont fournies dans le fichiers `data_tp7.json`. Ce fichier est chargés dans le script `Tp7_trilat.py` par la fonction `loadJson`. On obtient :

- PosSat : tableau numpy des corrdonnées ECEF des satellites
- Dobs : tableau nyumpy des distances
- X0 : tableau numpy des coordonnées approchées

`pygnsstoolbox` ...

La fonction de multilatération avec GPS seul est codée dans la bibliothèque `pygnssstoolbox` :

```
import gnsstoolbox.gnss_process as proc

dir_data = '../data'
PosSat, Dobs, X0 = loadJson(os.path.join(dir_data, "data_tp7
    .json"))

X,Y,Z,cdtr,sigma0_2,V,Qxx = proc.TrilatGps(PosSat,Dobs,X0)
print(X,Y,Z,cdtr,sigma0_2,V,Qxx)
```

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 8	Écriture d'une fonction de trilatération : II - moindres carrés pondérés	Difficulté : ⋆

Les observations à faible élévation seront moins précises à cause de l'erreur engendrée par le délai troposphérique. Il est ainsi possible de pondérer les observations en prenant $\sigma_{obs} = \frac{\sigma_{code}}{sin(ele)}$, où σ_{code} correspond à l'écart type sur une mesure de code ($\sigma_{code} \approx 2$ m) et ele correspond à l'élévation du satellite.

À programmer...

Modifier la fonction de multilatération afin de permettre une pondération des observations dans les moindres carrés.

```
dir_data = '../data'
PosSat, Dobs, X0, ElevSat = loadJson(os.path.join(dir_data,
    "data_tp8.json"))

X,Y,Z,cdtr,sigma0_2,V,SigmaX = estimation(PosSat,Dobs,
    ElevSat,X0)
print("\nSolution TP\nX = %13.3f m\nY = %13.3f m\nZ = %13.3f
     m\nc*dtr = %.9f m" % (X,Y,Z,cdtr))
print("sigma0_2 = %.3f" % (sigma0_2))
print("V = ",V,"\nSigmaX = ",SigmaX)
```

On calculera aussi les résidus normalisés : $V_{norm}(i) = \frac{V(i)}{\sigma_i}$.

Données test...

Afin de tester la fonction, un jeu de données test a été produit. On détermine la position du récepteur par multilatération à partir de 10 satellites. Les données sont fournies dans le fichiers `data_tp8.json`. Ce fichier est chargés dans le script `Tp8_trilat.py` par la fonction `loadJson`. On obtient :

— PosSat : tableau numpy des corrdonnées ECEF des satellites
— Dobs : tableau nyumpy des distances
— ElevSat : tableau numpy des élévations de satellites en radians
— X0 : tableau numpy des coordonnées approchées

pygnsstoolbox ...

La fonction de multilatération avec pondération des observation est codée dans la bibliothèque :

```
import gnsstoolbox.gnss_process as proc

dir_data = '../data'
PosSat, Dobs, X0, ElevSat = loadJson(os.path.join(dir_data,
    "data_tp8.json"))
satIndex = np.ones(Dobs.shape)

X,Y,Z,cdtr,cGGTP, cGPGL, sigma0_2,V,Qxx = proc.
    TrilatGnssPonderationElev(PosSat,Dobs,X0,satIndex,ElevSat
    )
print(X,Y,Z,cdtr,sigma0_2,V,Qxx)
```

La fonction `TrilatGnssPonderationElev` fournit à la fois un calcul GNSS et tenant compte de la pondération. Pour l'utiliser dans le contexte de ce TP, il convient de forcer satIndex à 1 pour tous les satellites, ce qui revient à dire que tous les satellites sont GPS.

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 9	Écriture d'une fonction de trilatération : III - multi-constellations	Difficulté : ⋆⋆⋆

Le système de temps est différent pour chaque constellation GNSS. Il existe ainsi un offset entre le système de temps des différentes constellations, qu'il peut être possible d'estimer.

On appellera ainsi $GGTO$ (GPS to Galileo Time Offset) l'offset entre le temps GPS et le temps Galileo, et $GPGL$ (GPS to GLonass time offset) l'offset entre le temps GPS et le temps Glonass. On considère pour ce deuxième offset, que le temps Glonass a été corrigé des secondes intercalaires (qui doivent être corrigées avant, sous peine de ne pas pouvoir calculer de position).

L'équation d'observation de la mesure de pseudo-disance (simplifiée) devient alors :

$$\begin{cases} PR^G_{corr} &= \sqrt{(X_{rec}-X^G_{sat})^2+(Y_{rec}-Y^G_{sat})^2+(Z_{rec}-Z^G_{sat})^2}+c\times dt_r \\ PR^R_{corr} &= \sqrt{(X_{rec}-X^R_{sat})^2+(Y_{rec}-Y^R_{sat})^2+(Z_{rec}-Z^R_{sat})^2}+c\times dt_r+c\times GPGL \\ PR^E_{corr} &= \sqrt{(X_{rec}-X^E_{sat})^2+(Y_{rec}-Y^E_{sat})^2+(Z_{rec}-Z^E_{sat})^2}+c\times dt_r+c\times GGTO \end{cases}$$

où :

- PR^G_{corr} : pseudo-distance mesurée entre un satellite GPS et le récepteur, corrigée
- PR^R_{corr} : pseudo-distance mesurée entre un satellite Glonass et le récepteur, corrigée
- PR^E_{corr} : pseudo-distance mesurée entre un satellite Galileo et le récepteur, corrigée
- $X_{rec}, Y_{rec}, Z_{rec}$: position du récepteur
- $X^G_{sat}, Y^G_{rec}, Z^G_{rec}$: position du satellite (G : GPS, R : Glonass, E : Galileo)
- dt_r : erreur d'horloge du récepteur
- $GPGL$: GPS to GLonass time offset
- $GGTO$: GPS to Galileo Time Offset
- c : célérité de la lumière dans le vide, $c = 299792458.0\,\mathrm{m.s^{-1}}$

À programmer...

Modifier la fonction de multilatération afin de permettre un calcul multi-constellations :

```
dir_data = '../data'
PosSat, Dobs, X0, ElevSat, SatIndex = loadJson(os.path.join(
    dir_data, "data_tp9.json"))

X,Y,Z,cdtr,sigma0_2,V,SigmaX = estimation(PosSat, Dobs,
    ElevSat, SatIndex, X0)
print("\nSolution TP\nX = %13.3f m\nY = %13.3f m\nZ = %13.3f
     m\nc*dtr = %.9f m" % (X,Y,Z,cdtr))
print("sigma0_2 = %.3f" % (sigma0_2))
print("V = ",V,"\nSigmaX = ",SigmaX)
```

En entrée de la fonction, ajouter le tableau satIndex (j x 1). Chaque ligne de ce tableau donne la constellation du satellite correspondant à la ligne des matrices $PosSat$ et $Dobs$. On pourra utiliser le code suivant : 1 pour GPS, 2 pour Galileo et 3 pour Glonass.

Les inconnues à estimer sont :

- les coordonnées du récepteur $(X_{rec}, Y_{rec}, Z_{rec})$
- l'erreur systématique effectuée sur toutes les distances due à l'erreur d'horloge récepteur $c \times dt_r$
- la distance engendrée par l'offset entre les systèmes de temps GPS et Galileo $c \times GGTO$
- la distance engendrée par l'offset entre les systèmes de temps GPS et Glonass $c \times GPGL$

Le vecteur $X0$ contient donc 6 valeurs approchées $[X; Y; Z; cdt_r; cGGTO; cGPGL]$. Les valeurs approchées de $GPGL$ et de $GGTO$ sont disponibles dans l'en-tête du fichier `RINEX` de navigation. Elles peuvent néanmoins être prises égales à 0 sans incidence pour le calcul.

On pondérera les observation dans un premier temsp de manière isotrope puis en tenant compte de l'élévations.

Attention : n'estimer $c \times GGTO$ que si suffisamment de satellites Galileo sont visibles (au moins 2).

Il peut être intéressant de permettre le calcul avec une constellation seule (GPS, Glonass ou bien Galileo). Dans ce cas, il n'y a pas d'offset (GPGL ou GGTO) à estimer. Le calcul est alors directement réalisé dans le système de temps de la constellation et non dans celui

du GPS.

Données test...

Afin de tester la fonction, un jeu de données test a été produit. On détermine la position du récepteur par multilatération à partir de 10 satellites. Les données sont fournies dans le fichiers `data_tp9.json`. Ce fichier est chargés dans le script `Tp9_trilat.py` par la fonction `loadJson`. On obtient :

- PosSat : tableau numpy des corrdonnées ECEF des satellites
- Dobs : tableau nyumpy des distances
- ElevSat : tableau numpy des élévations de satellites en radians
- SatIndex : tableau numpy d'identifiants de constellation
- X0 : tableau numpy des coordonnées approchées

pygnsstoolbox ...

La fonction de multilatération multi-constellations est codée dans la bibliothèque

```
import gnsstoolbox.gnss_process as proc

    dir_data = '../data'
PosSat, Dobs, X0, ElevSat, SatIndex = loadJson(os.path.join(
    dir_data, "data_tp9.json"))

X,Y,Z,cdtr,cGGTP, cGPGL, sigma0_2,V,SigmaX = proc.
    trilatGnssPonderationElev(PosSat,Dobs,X0,satIndex,ElevSat
    )
print("\nSolution pygnssToolbox\nX = %13.3f m\nY = %13.3f m\
    nZ = %13.3f m\nc*dtr = %.9f m" % (X,Y,Z,cdtr))
print("s0 = ", sigma0_2,"\nV = ", V)
print("SigmaX", SigmaX)
```

Remarque...

Le calcul est pour l'instant réalisé dans le système de temps GPS. Les horloges atomiques définissant le temps Galilo étant de meilleures qualité que les horloges GPS, il sera judicieux de réaliser le calcul dans le système de temps Galileo lorsque la constellation sera complète.

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP — Université de Technologie Ouverte Pluripartenaire
Partie : 10	Positionnement GPS sur le code : I - premier calcul	Difficulté : ⋆⋆

Reprenons l'équation d'observation de la mesure de pseudo-disance dans le cas du GPS seul :

$$PR = \rho + c \times dt_r - c \times dt_e - c \times dt_{relat} + d_{tropo} + d_{iono}$$

où

- PR : pseudo-distance mesurée (code C/A ou code P)
- ρ : distance géométrique
- dt_r : erreur d'horloge du récepteur
- dt_e : erreur d'horloge du satellite
- dt_{relat} : correction relativiste
- d_{tropo} : retard troposphérique
- d_{iono} : retard ionosphérique
- c : célérité de la lumière dans le vide $c = 299792458.0\,\mathrm{m.\,s^{-1}}$

Dans cette partie vont être calculées les différentes corrections à appliquer aux mesures de pseudo-distances, ainsi que la position des satellites à chaque époque.

Dans un premier temps, on se limitera aux erreurs d'horloges récepteur et satellite, à la correction relativiste ainsi que la rotation de la terre pendant la durée de propagation du signal. Une fois ce premier calcul réalisé, il sera alors possible d'ajouter les corrections troposphériques, ionosphériques, la hauteur d'antenne , puis les constellations Glonass et Galileo (TD 11 et 12).

⌨ À programmer...

```
""" lecture du fichier BRDC """
Orb = orbits.orbit()
```

```
dir_orb = '../data'
Orb.loadRinexN(os.path.join(dir_orb, '
    BRDC00IGN_R_20212400000_01D_MN.rnx'))

""" Definition de l'instant pour lequel on cherche une
    position """
t = gpst.gpsdatetime(yyyy=2021,doy=240,dsec=5400)
print(t)

rnx = rx.rinex_o()
filename = os.path.join(dir_orb,'
    MLVL00FRA_R_20212400000_01D_30S_MO.21o')
ret = rnx.loadRinexO(filename)

if ret<0:
    print(ret)
    return

Ep = rnx.getEpochByMjd(t.mjd)

spp1 = gnss_process_TP()
spp1.const='G'
spp1.constraint=0
spp1.cut_off = 10 * d2r
spp1.X0[0]=rnx.headers[0].X
spp1.X0[1]=rnx.headers[0].Y
spp1.X0[2]=rnx.headers[0].Z
spp1.spp(Ep,Orb)

spp2 = proc.gnss_process()
spp2.const='G'

print("Calcul sur les orbites brdc")
Ep2 = spp2.spp(Ep,Orb)
print("X = %.2fm Y = %.2fm Z = %.2fm " % (Ep2.X, Ep2.Y, Ep2.
    Z))
#
print("PRN %-16s %-16s %-16s %-16s" % ('Xs', 'Ys', 'Zs', 'PR
    '))
for s in Ep2.satellites:
```

```
    print("%1s%02d %16.3f %16.3f %16.3f %16.3f" % (s.const,s
    .PRN, s.Xs, s.Ys, s.Zs, s.PR))
print("V",Ep2.V)
```

Qui prenne en entrée :

- Ep : une époque Rinex
- Orb : message de navigation

Et calcule en sortie :

- X,Y,Z : coordonnées du récepteur
- $c \times dt_r$

Récupération des observations et éphémérides pour une époque donnée

Dans un premier temps on calculera la position du récepteur à partir d'observations GPS sur le code C1.

À programmer...

Récupérer les observations GPS pour une époque donnée en entrée de la fonction.

Les pseudo-distances sont contenues dans les champs $C1$ et $P2$.

Attention à vérifier la présence d'éphémérides (ou arcs d'orbites précises) pour les satellites observés. Seules les observations pour lesquelles il sera possible de calculer la position du satellite doivent être prises en compte.

Calcul de la position des satellites à partir de l'instant de réception et du temps de vol de l'onde

Durant le temps de vol de l'onde, les satellites continuent à se déplacer. La position dont nous avons besoin est celle du satellite au moment du "départ" de l'onde.

À programmer...

On utilisera la pseudo-distance observée. Le temps de propagation sera calculé avec la relation

$$Temps\_de\_vol = \frac{PR}{c}$$

avec $c = 299792458.0\,\mathrm{m.s^{-1}}$

Calculer la date d'émission du signal.

Correction de la dérive d'horloge du satellite

L'horloge embarquée à bord des satellites GPS n'est pas parfaitement calée sur l'horloge centrale du système GPS et donc sur le temps GPS. Il convient donc de corriger les observations de cette erreur.

Suivant le type d'éphémérides utilisées pour le calcul -radiodiffusées ou précises-, 2 algorithmes différents peuvent être utilisés :

1. Algorithme utilisant des éphémérides radiodiffusées

   A partir des valeurs fournies dans le fichier `RINEX` de navigation, déterminer, pour chaque satellite, la dérive d'horloge du satellite par rapport au temps GPS

   $$dt_e = \alpha_0 + \alpha_1 \times (t - TOC) + \alpha_2 \times (t - TOC)^2$$

   Attention à bien spécifier t dans la même échelle de temps que TOC (Time Of Clock) à récupérer dans le message de navigation. α_0, α_1, α_2 sont aussi fournis dans le message de navigation.

2. Utilisation des éphémérides précises

   Dans ce cas il suffit d'utiliser le polynôme de Lagrange défini pour l'interpolation des positions appliqué cette fois aux erreurs d'horloge.

À programmer...

Calculer l'erreur d'horloge de chaque satellite.

Attention, cette correction s'applique sur deux termes :

1. l'heure d'émission du signal qui sert à calculer la position des satellites (-dt_e)
2. la pseudo-distance. $c \times dt_e$ apparait comme une correction à la pseudo-distance observée (+ $c \times dt_e$).

Effet relativiste

Le calcul de la position du récepteur est impacté par des effets relativistes. Suivant le type d'éphémérides utilisées pour le calcul - radiodiffusées ou précises -, 2 algorithmes différents

peuvent être utilisés :

1. Algorithme utilisant des éphémérides radiodiffusées :

$$\Delta t_{relat} = F \times e \times \sqrt{a} \times \sin E_k$$

où a est le demi-grand axe de l'ellipse décrivant la trajectoire du satellite, e son excentricité, E_k l'anomalie excentrique et F une constante ($F = -4.442807633e^{-10}$).

2. Algorithme utilisant des éphémérides précises :

$$\Delta t_{relat} = -2\frac{\overrightarrow{r}.\overrightarrow{v}}{c^2}$$

où
— $\overrightarrow{r}$: vecteur position instantané du satellite
— $\overrightarrow{v}$: vecteur vitesse instantanée du satellite
$\overrightarrow{r}$ et $\overrightarrow{v}$ peuvent être exprimés indifféremment dans un repère inertiel ou dans un repère tournant.

À programmer...

Calculer la correction relativiste pour chaque satellite.

Comme pour l'erreur d'horloge satellite, cette correction doit être appliquée :

1. à la date d'émission du signal pour le calcul des positions des satellites
2. comme un terme correctif à la pseudo-distance

Calcul de la position des satellites

Maintenant que l'instant d'émission a été calculé, il est possible de déterminer la position des satellites.

À programmer...

Calculer la position de chaque satellite à l'instant d'émission du signal GPS.

Et pourtant elle tourne...

Si un calcul GPS est lancé à ce stade, il doit rester un écart important en longitude.

En fait, le choix du type d'orbites -radiodiffusées ou précises- induit le système de référence dans lequel elles sont exprimées. Cette "bizarrerie" est liée aux services impliqués dans l'estimation de ces orbites. Les éphémérides GPS radiodiffusées sont exprimées dans le référentiel WGS84 alors que les orbites précises le sont dans le référentiel IGS14. Dans le cas qui nous concerne, cela n'a que peu d'influence sur le résultat. Rappelons que ces deux référentiels sont des repères géocentriques tournant avec la terre (Earth Centered-Earth Fixed). Comme la terre a tourné pendant le temps de trajet, l'intersection calculée n'a que peu de sens puisque calculée dans autant de référentiels différents qu'il y a de satellites à des distances différentes de la station.

 À programmer...

La méthode revient à faire subir à chaque satellite une rotation correspondant à son temps de parcours de façon à ce que tous les satellites aient des coordonnées exprimées dans le référentiel du temps de réception. Dans ce cas l'intersection spatiale a un sens.

Premier calcul de position

A ce stade du calcul, vous devez disposer :

- de la position des satellites
- des pseudo-distances corrigées

Il est alors possible de calculer la position du récepteur dans le référentiel WGS84 !

 À programmer...

Reprendre le script et calculer la position du récepteur de la station à chaque époque en utilisant les fonctions de prétraitements et de multilatération précédemment développées.

Vous venez de réaliser un GPS de navigation.. !

ENSG Géomatique ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 11	Positionnement GPS sur le code : II - corrections supplémentaires	Difficulté : ⋆⋆

La position calculée dans le TD 10 est celle du centre de phase du récepteur. Afin de déterminer la position du point mesuré, il faut au minimum prendre en compte la hauteur d'antenne. Celle-ci est présente dans l'en-tête des fichiers d'observation `RINEX` des stations du RGP. La calibration d'antenne ne sera pas prise en compte dans ce TD.

Ajout d'un angle de coupure

Pour de faibles élévations, la couche d'atmosphère traversée par les signaux GNSS est très importante ce qui génère un délai troposphérique élevé. Ce délai est alors mal modélisé, ce qui peut causer des erreurs supplémentaire. L'idée est de supprimer les observations des satellites dont l'élévation est trop faible.

Il faut donc dans un premier temps déterminer l'azimut et l'élévation de chaque satellite. Il est nécessaire à ce stade de disposer de coordonnées approchées suffisamment précises. On va donc calculer la position du satellite dans un repère local centré sur le point d'observation et dont l'axe Z correspond à la verticale locale. Il s'agit en fait de la normale à l'ellipsoïde qui est assimilée à la verticale étant donné la précision demandée.

La formule de changement de repère est alors

$$\begin{bmatrix} X_{sat} \\ Y_{sat} \\ Z_{sat} \end{bmatrix}_{local} = \begin{bmatrix} -\sin\lambda_1 & \cos\lambda_1 & 0 \\ -\sin\varphi_1\cos\lambda_1 & -\sin\varphi_1\sin\lambda_1 & \cos\varphi_1 \\ \cos\varphi_1\cos\lambda_1 & \cos\varphi_1\sin\lambda_1 & \sin\varphi_1 \end{bmatrix} \begin{bmatrix} X_{sat} - X_1 \\ Y_{sat} - Y_1 \\ Z_{sat} - Z_1 \end{bmatrix}_{WGS84}$$

où (X_1, Y_1, Z_1) et (λ_1, φ_1) sont les coordonnées cartésiennes et géographiques approchées de la station dans le système WGS84.

Connaissant les coordonnées de chaque satellite dans le repère local, on peut calculer leur azimut et leur élévation par trigonométrie classique dans le plan horizontal pour l'azimut et dans le plan vertical pour la distance zénithale.

À programmer...
Pour chaque satellite, calculer son élévation et ne pas prendre en compte les observations des satellites dont l'élévation est inférieure à 7°.

Pour transformer les coordonnées cartésiennes approchées en coordonnées géographiques, il est possible d'utiliser la fonction `tools.toolCartGeoGRS80(X, Y, Z)`.

`pygnsstoolbox` ...

L'azimut et l'élévation des satellites, ainsi que la hauteur ellipsoidale du récepteur sont donnés par :

```
import gnsstoolbox.gnsstools as tools
[az,ele,h] = tools.toolAzEleH(X,Y,Z,Xs,Ys,Zs)

```

Retard troposphérique

Il est possible de corriger les observations du retard troposphérique en utilisant un modèle, par exemple le modèle de Saastamoinen qui corrige assez bien la partie hydrostatique du retard.

$$d_{tropo} = \frac{0,002277}{\cos Z}\left[P + \left(\frac{1255}{T} + 0,05\right)e - B\tan^2 Z\right] + \delta R$$

où

- Z : distance zénithale du satellite
- T : température au niveau de la station en Kelvin
- P : pression atmosphérique au niveau de la station en hPa
- e : pression partielle de vapeur d'eau en hPa
- B : terme correctif dépendant de H, hauteur de la station
- δR : terme correctif dépendant de la heuteur et de la zénithale

$$e = \frac{R_h}{100}\exp\left(-37,2465 + 0,213166T - 0,000256908T^2\right)$$

où

- R_h : humidité relative en %

Lors des calculs, on prendra

— $T = 285\,\mathrm{K}$

— $P = 1013,15\,\mathrm{hPa}$

— $R_h = 50\%$

Terme correctif $B(h)$:

Hauteur km)	B (hPa)
0,0	1,156
0,5	1,079
1,0	1,006
1,5	0,938
2,0	0,874
2,5	0,813
3,0	0,757
4,0	0,654
5,0	0,563

Terme correctif $\delta R(Z, h)$:

h	Z												
	60	66	70	73	75	76	77	78	78,5	79	79,5	79,75	80
0,0	0,003	0,006	0,012	0,020	0,031	0,039	0,050	0,065	0,075	0,087	0,102	0,111	0,121
0,5	0,003	0,006	0,011	0,018	0,028	0,035	0,045	0,059	0,068	0,079	0,093	0,101	0,110
1,0	0,002	0,005	0,010	0,017	0,025	0,032	0,041	0,054	0,062	0,072	0,085	0,092	0,100
1,5	0,002	0,005	0,009	0,015	0,023	0,029	0,037	0,049	0,056	0,065	0,077	0,083	0,091
2,0	0,002	0,004	0,008	0,013	0,021	0,026	0,033	0,044	0,051	0,059	0,070	0,076	0,083
3,0	0,002	0,003	0,006	0,011	0,017	0,021	0,027	0,036	0,042	0,049	0,058	0,063	0,068
4,0	0,001	0,003	0,005	0,009	0,014	0,017	0,022	0,030	0,034	0,040	0,047	0,052	0,056
5,0	0,001	0,002	0,004	0,007	0,011	0,014	0,018	0,024	0,028	0,033	0,039	0,043	0,047

À programmer...

Calculer le retard troposphérique pour chaque mesure de pseudo-distance. Afin de simplifier l'écriture du programme et compte-tenu de l'amplitude des corrections, on néglige les termes δR ainsi que $B \tan^2 Z$.

La relation à programmer devient donc :

$$\delta = \frac{0,002277}{\cos Z}\left[P + \left(\frac{1255}{T} + 0,05\right)e\right]$$

`pygpstoolbox` ...

La correction troposphérique est donnée par :

```
import gnsstoolbox.gnss_corr as corr
[dr]=corr.corr_dtropo_saast(P,T,H,h,zen)

```

où $zen = \frac{\pi}{2} - ele$.

Corriger ainsi la pseudo-distance du délai troposphérique : $PR_{corr} = PR - d_{tropo}$, juste avant la correction de la rotation de la Terre.

Effet ionosphérique

Suivant les observations disponibles, deux algorithmes peuvent être utilisés :

1. Données monofréquence (optionnel, l'algorithme est long à coder)

   Dans ce cas, nous allons devoir utiliser un modèle de correction ionosphérique. Nous allons utiliser le modèle de Klobuchar [Klobuchar, 1975]. Ce modèle donne le retard pour chaque satellite en fonction de son azimut et de son élévation.

$$d_{iono} = c \times \begin{cases} F * \left[5.0 * 10^{-9} + AMP * \left(1 - \frac{x^2}{2} + \frac{x^4}{24}\right)\right] & pour\ |x| < 1.57 \\ F * 5.0 * 10^{-9} & pour\ |x| \geq 1.57 \end{cases}$$

$$AMP = \begin{cases} \sum_{n=0}^{3} \alpha_n \phi_m^n & si\ AMP \geq 0 \\ 0 & si\ AMP < 0 \end{cases}$$

   Les α_n sont fournis dans le fichier `RINEX` de navigation (champ $GPSA$ de NAV_header).

$$x = \frac{2\pi * (t - 50400)}{PER}$$

   avec $t = 43200 * \lambda_i + t_{GPS}$ en secondes (secondes dans le jour). t doit être compris entre 0 et 86400 (effectuer eventuellement un modulo 86400).

$$PER = \begin{cases} \sum_{n=0}^{3} \beta_n \phi_m^n & si\ PER \geq 72.000 \\ 72.000 & si\ PER < 72.000 \end{cases}$$

   Les β_n sont fournis dans le fichier `RINEX` de navigation (champ $GPSB$ de NAV_header).

   AMP et PER sont donnés en secondes.

ϕ_m correspond à la latitude géomagnétique de la projection terrestre du point d'intersection ionosphérique.

$$\phi_m = \phi_i + 0.064 * \cos(\lambda_i - 1.617)$$

ϕ_i, λ_i sont les coordonnées géodésiques de la projection terrestre du point d'intersection ionosphérique.

$$\lambda_i = \lambda_u + \frac{\psi \sin A}{\cos \phi_i}$$

λ_u représente la longitude de la station, ψ l'angle au centre entre la station et la projection terrestre du point d'intersection ionosphérique et A l'azimut du satellite.

$$\psi = \frac{0.0137}{E + 0.11} - 0.022$$

$$\phi = \begin{cases} \phi_u + \psi \cos A & si\, |\phi_i| \leq 0.416 \\ 0.416 & si\, \phi_i \geq 0.416 \\ -0.416 & si\, \phi_i \leq -0.416 \end{cases}$$

E étant l'élévation du satellite et ϕ_u la latitude le la station.

Il reste à calculer le facteur d'obliquité. Il s'agit de l'influence de l'obliquité sur le retard ionosphérique.

$$F = 1.0 + 16.0 * (0.53 - E)^3$$

Les modèles ci-dessus sont calculés à partir des deux fréquences. Comme on n'utilise qu'une seule fréquence, on doit corriger les observations d'une valeur appelée TGD fournie dans le message de navigation ($Eph.TGD$). Le TGD est ajouté au temps d'émission et $c \times TGD$ est soustrait à la pseudo-distance.

Corriger ainsi la pseudo-distance du délai troposphérique et du TGD :

$$PR_{corr} = PR - d_{iono} - c \times TGD$$

juste avant la correction de la rotation de la Terre.

2. Données bifréquences

   Le document IS-GPS-200 donne un algorithme de calcul d'une pseudo-distance dite iono-free.

$$P_3 = \rho_{ionofree} = \frac{C_2 - \gamma C_1}{1 - \gamma}$$

avec $\gamma = (\frac{f_{L1}}{f_{L2}})^2$ avec f_{L1} = 1575.42 MHz et f_{L2} = 1227.60 MHz

On peut alors refaire tout le calcul en remplaçant le code C/A utilisé depuis le début par $\rho_{ionofree}$.

À programmer...

Créer une fonction `[iono_free] = bifreq(C1, C2)` qui calcule la combinaison iono-free à partir des observations sur C1 et C2. Relancer le calcul en remplaçant les observations sur C1 par les observations sur P3.

ENSG Géomatique ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 12	Positionnement GNSS sur le code : III - prise en compte des observations Glonass et Galileo	Difficulté : ⋆⋆⋆

Á ce stade vous pouvez calculer la position du récepteur dans le référentiel WGS84 à chaque époque à partir d'observations GPS. Dans cette partie nous allons ajouter les observations Glonass et Galileo.

> Remarque...
> Les orbites radiodiffusées des constellations GPS, Glonass et Galileo ne sont pas exprimées dans le même référentiel. Cependant nous considérerons qu'elles sont quand même dans un même référentiel, l'erreur engendrée par cette approximation (≈ centimétrique à décimétrique) étant faible par rapport au bruit de mesure sur le code (≈ 1 mètre).

Observations

Les observations de pseudo-distance sont contenues dans les champs $C1$ et $C2$ de la structure Obs. Ceci est valable pour toutes les constellations dans le cadre de ce TD (ainsi C5 est emplacé par C2 pour Galileo).

Correction ionosphérique

Le modèle de Klobuchar est calculé pour la fréquence $f1$ du GPS. Elle peut être appliquée à une autre fréquence fc (du GPS ou d'autres constellations) en utilisant la formule suivante :

$$diono_{fc} = diono_{f1_{GPS}} \times \left(\frac{f1_{GPS}}{fc}\right)^2$$

Avec $f1$ = 1575.42 MHz.

On a les fréquences suivantes :

- GPS : $f1_{GPS}$ = 1575.42 MHz, $f2_{GPS}$ = 1227.60 MHz
- Glonass : $f1_{GLO} = 1602.00 + freq\_num \times 0.5625$ et $f2_{GLO} = 1246.00 + freq\_num \times 0.4375$. $freq\_num$ est un offset donné dans l'éphéméride radiodiffusée de Glonass

— Galileo : $f1_{GAL} = f1_{GPS}$ et $f5_{GAL}$ = 1176.45 MHz

Pour Galileo, le TGD est contenu dans le champ $BGDE5a$ des éphémérides radiodiffusées. Il n'y a pas de TGD radiodiffusé pour Glonass.

De même, pour la combinaison $iono-free$:

$$P_3 = \rho_{ionofree} = \frac{C_2 - \gamma C_1}{1-\gamma}$$

avec $\gamma = (\frac{f_1}{f_2})^2$. f_1 et f_2 correspondent aux fréquences des observations C1 et C2.

Secondes intercalaires

Le temps Glonass étant lié au temps UTC (et non au temps TAI comme GPS et Galileo), le temps d'émission des observation Glonass doit être corrigé du nombre de secondes intercalaires donné dans le champ $LEAP\_SECONDS$ de la classe NAV. Cette correction doit être appliquée avant toutes les autres corrections (avant le calcul du temps de vol). Elle n'a pas à être réalisée si des éphémérides précises sont utilisées.

Erreur d'horloge satellite

L'erreur d'horloge satellite dt_e des satellites Glonass est calculée par :

$$\begin{aligned} t_0 &= t - TOC \\ dt_e &= SV\_clock\_offset + SV\_relat\_freq\_offset \times t_0 \end{aligned}$$

avec SV_clock_offset et $SV\_relat\_freq\_offset$ donnés dans le message de navigation Glonass.

Erreur relativiste

Les orbites Glonass étant calculées par intégration numérique, il est possible d'utiliser les valeurs Vx, Vy et Vz calculées par intégration numérique pour calculer dt_{relat} pour les satellites Glonass. On applique la même formule que pour le calcul de dt_{relat} avec des orbites précises.

À programmer...
Modifier la fonction `spp()` afin de prendre en compte les constellations Glonass et Galileo. Modifier ensuite le script afin de calculer la position du récepteur en tenant compte de plusieurs constellations.

Calcul des DOP

Les DOP's (Dilution Of Precision) sont estimés à partir de la matrice de variance covariance obtenue lors d'un calcul SPP avec une pondération unitaire.

A partir des positions des satellites et des coordonnées du récepteur (éventuellement approchées), poser la matrice modèle A.

On aura alors :

$N = A^t A$

$\Sigma_{\hat{X}} = N^{-1}$

$$\Sigma_{\hat{T}} = R \times \Sigma_{\hat{X}} \times R^t$$

où R est la matrice de passage permettant d'exprimer la matrice modèle dans le repère local et $\Sigma_{\hat{X}}$ la matrice de variance-covariance sur les paramètres estimés (dans le référentiel cartésien géocentrique).

$$R = \begin{pmatrix} -\sin\lambda & \cos\lambda & 0 & 0 \\ -\sin\varphi\cos\lambda & -\sin\varphi\sin\lambda & \cos\varphi & 0 \\ \cos\varphi\cos\lambda & \cos\varphi\sin\lambda & \sin\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

où λ et φ sont les coordonnées géographiques du récepteur.

Par un calcul de propagation de la variance à partir de la matrice de variance-covariance du système (avec une pondération unitaire), calculer :

- GDOP = $\sqrt{\sigma_E^2 + \sigma_N^2 + \sigma_h^2 + \sigma_t^2}$
- PDOP = $\sqrt{\sigma_E^2 + \sigma_N^2 + \sigma_h^2}$
- HDOP = $\sqrt{\sigma_E^2 + \sigma_N^2}$
- VDOP = $\sqrt{\sigma_h^2}$
- TDOP = $\sqrt{\sigma_t^2}$

Calcul estimateurs de précision du calcul

Contrairement au calcul précédent, il s'agit ici non plus d'estimer la qualité de la géométrie de la constellation mais bien la qualité du calcul réalisé, tenant compte des erreurs d'observations.

On calculera donc :

- les écarts-types sur les coordonnées locales estimées
- la matrice de corrélation associée au système

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 14	Calcul d'une position à partir des mesures de code en DGNSS	Difficulté : ⋆⋆⋆

Principe du traitement DGNSS

Le DGNSS (*Differential GNSS*) est une technique de positionnement différentiel sur le code. Ce mode de positionnement est plus précis car il permet de minimiser les erreurs liées à la précision des orbites, des horloges satellite, ainsi que les erreurs liées à la propagation des ondes électromagnétiques dans l'atmosphère.

Il sera ainsi nécessaire de disposer de deux stations qui observent simultanément. Les coordonnées d'une station doivent être connues (station de base), les coordonnées de la seconde station (station mobile) devant être estimées.

Dans un premier temps, les corrections PRC (*PseudoRange Correction*) sont calculées à partir des données acquises par la station de base. Elles sont ensuites appliquées aux observations de la seconde station.

Après cette correction, il demeure principalement le bruit de mesure.

Cette technique de correction est d'autant plus efficace que la distance entre les deux stations est faible. Elle est efficace jusqu'à une centaine de kilomètres entre les stations.

La précision ainsi atteinte est submétrique.

Mise en équations

Reprenons l'équation d'observation de la meseure de pseudo-distance dans le cas du GPS seul, pour un récepteur i et un satellite j :

$$PR_i^j = \rho_i^j + d\rho_i^j + c \times dt_{r,i} - c \times dt_e^j - c \times dt_{relat}^j + d_{tropo,i}^j + d_{iono,i}^j$$

où

— PR_i^j : pseudo-distance mesurée entre le récepteur i et un satellite j

— $\rho_i^j = \sqrt{(X_i - X^j)^2 + (Y_i - Y^j)^2 + (Z_i - Z^j)^2}$: distance géométrique entre le récep-

teur i et un satellite j

— $d\rho_i^j$: erreur sur l'orbite du satellite j

— $dt_{r,i}$: erreur d'horloge du récepteur i

— dt_e^j : erreur d'horloge du satellite j

— dt_{relat}^j : correction relativiste du satellite j

— $d_{tropo,i}^j$: retard troposphérique

— $d_{iono,i}^j$: retard ionosphérique

— c : célérité de la lumière dans le vide $c = 299792458.0\,\mathrm{m.s^{-1}}$

Le terme $d\rho_i^j$ rajouté correspond à l'erreur sur l'orbite. Si la station i est fixe, de coordonnées connues, le terme ρ_i^j est alors connu. Le PRC peut ainsi être calculé :

$$\begin{cases} PRC^j &= \rho_i^j + c \times dt_{r,i} - c \times dt_e^j - c \times dt_{relat}^j + d_{tropo,i}^j + d_{iono,i}^j - PR_i^j \\ &= -d\rho_i^j + err\_model\_dt_e^j - err\_model\_tropo_i^j - err\_model\_iono_i^j \end{cases}$$

où

— PRC : pseudorange correction

— $d\rho_i^j$: erreur sur l'orbite du satellite présente dans les éphémérides

— $err\_model\_dt_e^j$: erreur sur la modélisation de l'horloge satellite présente dans les éphémérides

— $err\_model\_tropo_i^j$: erreur de modélisation de la troposphère

— $err\_model\_iono_i^j$: erreur de modélisation de la ionosphère

Le PRC modélise ainsi les différentes erreurs affectant le calcul d'une position.

Remarque ...

On peut remarquer que $PRC^j = -V^j$, où V^j correspond au résidu du calcul par moindres carrés de $c \times dr_{r,i}$, pour l'observation du satellite j.

En effet, nous avons :

$$\underbrace{PR_i^j - \rho_i^j + c \times dt_e^j + c \times dt_{relat}^j - d_{tropo,i}^j - d_{iono,i}^j}_{variables\ connues\ ou\ modelisees} = \underbrace{c \times dt_{r,i}}_{a\ estimer}$$

Soit :

$$PR^j_{corr,i} = c \times dt_{r,i} \Leftrightarrow B = AX$$

avec :

$$B = \begin{bmatrix} PR^{j1}_{corr,i} \\ \vdots \\ PR^{jn}_{corr,i} \end{bmatrix}, A = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}, X = [cdt_r]$$

d'où

$$V = B - AX = -PRC$$

Pour un second récepteur, k, proche de i, la pseudo-distance corrigée est donnée par :

$$\begin{cases} PR^j_{corr,k} &= PR^j_k + PRC^j \\ &= \rho^j_k + (d\rho^j_i - d\rho^j_k) - c \times dt_{r,i} + (c \times dt^j_e - err\_model\_dt^j_e) + c \times dt^j_{relat} \\ &\quad -(d^j_{tropo,k} - err\_model\_tropo^j_i) - (d^j_{iono,k} - err\_model\_iono^j_i) \end{cases}$$

Si i et k sont proches :

- $d\rho^j_i - d\rho^j_k \approx 0$: les erreurs d'orbites sont corrigées
- $c \times dt^j_e - err\_model\_dt^j_e \approx$ erreur d'horloge 'vraie' : l'erreur d'horloge satellite est mieux modélisée
- $d^j_{tropo,k} - err\_model\_tropo^j_i \approx$ erreur troposphérique 'vraie' : le délai troposphérique est mieux modélisé
- $d^j_{iono,k} - err\_model\_iono^j_i \approx$ erreur ionosphérique 'vraie' : le délai ionosphérique est mieux modélisé

Il subsiste alors le bruit de mesure qui est la principale source d'erreurs.

Calcul de position par DGNSS

Pour calculer la position du récepteur mobile par DGNSS, nous allons procéder en 4 étapes. Pour chaque époque il faudra :

- récupérer les observations communes aux deux stations
- calculer les corrections PRC à partir des données de la station fixe
- corriger les pseudo-distances de la station mobile

— calculer la position du recepteur mobile

Nous pouvons ainsi nous baser sur la fonction `pretraitements_spp()` pour réaliser les 3 premières étapes, et sur la fonction `trilat()` pour réaliser les deuxième et quatrième étapes.

Le calcul devra être réalisé sur les mêmes observations (C1, C2 ou combinaison iono-free) entre les deux stations afin que la correction PRC ait un sens.

`pygpstoolbox` ...

La fonction réalisant les prétraitements nécessaires dans le cadre d'un positionnement DGNSS est :

```
interf_calc_preprocessing_DGNSS()
```

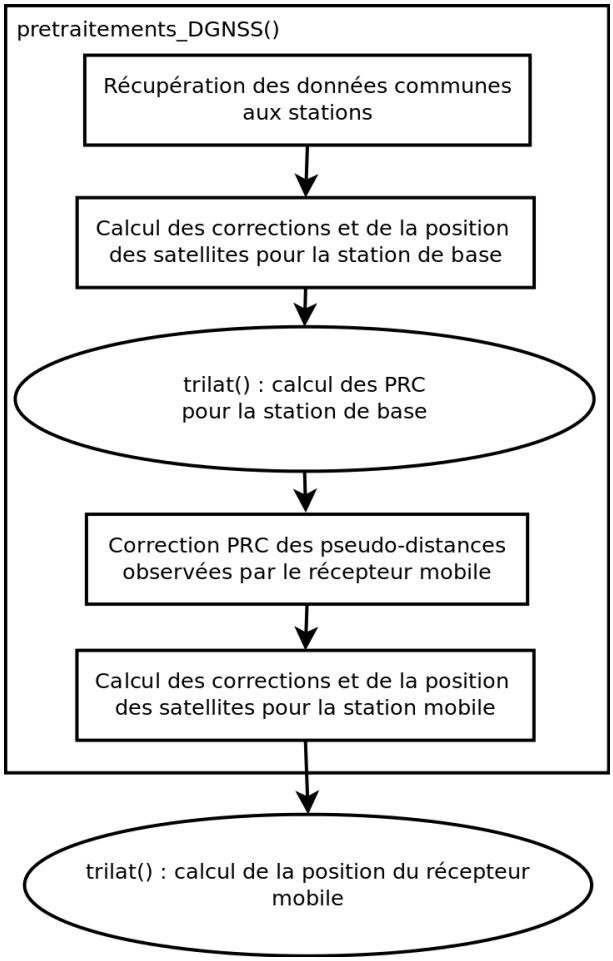


FIG 8 - Opérations à réaliser dans les fonctions `pretraitements_DGNSS()` et `trilat()`

Modification de la fonction `trilat()` afin de permettre le calcul des corrections PRC

Nous avons remarqué que la correction PRC est en réalité égale à $-V$ (opposé des résidus) dans le cadre d'une estimation par moindres carrés de l'erreur d'horloge récepteur dr_r, avec la position du récepteur fixe.

Nous allons procéder de manière légèrement différente : au lieu de fixer les coordonnées de la station de base, nous allons les estimer et ajouter une contrainte dans le calcul. La position de la station de base estimée sera ainsi contrainte à la position officielle/connue de la station. La contrainte peut être fixée à σ_c = 2 cm, ce qui est équivalent à dire que :

$$Pos_{rec} = Pos_{rec\ fixee} \pm 0.02m$$

Nous avons ainsi le système d'équations suivant :

$$\begin{cases} PR_{corr} &= \sqrt{(X_{rec} - X_{sat})^2 + (Y_{rec} - Y_{sat})^2 + (Z_{rec} - Z_{sat})^2} + c \times dt_r \\ X_{rec} &= X_{rec\ fixee} \\ Y_{rec} &= Y_{rec\ fixee} \\ Z_{rec} &= Z_{rec\ fixee} \end{cases}$$

Avec pour matrice de pondération :

$$P = \begin{pmatrix} P_{init} & \cdots & \cdots & \cdots \\ \vdots & \frac{1}{\sigma_c^2} & 0 & 0 \\ \vdots & 0 & \frac{1}{\sigma_c^2} & 0 \\ \vdots & 0 & 0 & \frac{1}{\sigma_c^2} \end{pmatrix}$$

où P_{init} correspond à la matrice de pondération du système sans contraintes.

 À programmer...

Modifier la fonction `trilat()` afin de permettre l'ajout de contraintes sur la position.

```
[X, Y, Z, cdtr, cGGTO, cGPGL, V, Vnorm, sigma02, Qxx, sat_const,
sat_PRN] = trilat(PosSat, Dobs, ElevSat, sat_const, sat_PRN, X0,
contrainte )
```

Cette fonction doit prendre en entrée :

- PosSat : position des satellites [X, Y, Z] en mètres
- Dobs : pseudo-distances corrigées, en mètres

— ElevSat : élévation des satellites en radians

— sat_const : indice de la constellation des satellites (1 : GPS, 2 : Galileo, 3 : Glonass)

— sat_PRN : PRN des satellites

— X0 : coordonnées de l'antenne de la station de base, suivies par les valeurs approchées de dt_r, $GGTO$ et $GPGL$: $[X; Y; Z; cdt_r; cGGTO; cGGPL]$

— contrainte : si égal à 1 : ajout d'une contrainte sur les positions

Et sont calculés en sortie :

— X, Y, Z : la position de la station

— cdtr, cGGTO, cGPGL : l'erreur d'horloge récepteur, ainsi que les offsets entre les différents systèmes de temps

— V, Vnorm : les résidus et résidus normalisés

— sigma02 : le facteur unitaire de variance au carré

— Qxx : la matrice de variance-covariance

— sat_const, sat_PRN : l'indice de la constellation des satellites utilisés dans l'estimation par moindres carrés, ainsi que leur PRN

Les paramètres sat_const et sat_PRN donnés en sortie permettront de définir pour quel satellite correspond chaque PRC calculé.

Fonction `pretraitements_DGNSS()`

La fonction `pretraitements_DGNSS()` va remplacer la fonction `pretraitements_spp()` afin de prendre en compte les corrections PRC. Les étapes à réaliser dans cette fonction sont les suivantes :

1. Récupération des observations communes aux récepteurs de base et mobile, pour un instant donné
2. Calcul des corrections et de la position des satellites pour la station de base uniquement
3. Calcul des PRC, avec la fonction `trilat()`
4. Correction des observations de la station mobile
5. Calcul des corrections et de la position des satellites pour la station mobile

Les étapes 2 et 5 ont déjà été ralisées dans la fonction `pretraitements_spp()`, il reste à

développer les 3 autres étapes.

À programmer...
Écrire la fonction :
`[Pos_sat, Dobs_corr, elev_sat, sat_const, sat_PRN] = pretraitements_DGNSS( RNX_header_base, RNX_data_base, RNX_header_mobile, RNX_data_mobile, NAV_header, NAV_data ,epoque, X0_base, X0_mobile )`

Cette fonction doit prendre en entrée :

- les observations `RINEX` de la station de base (RNX_header_base, RNX_data_base)
- les observations `RINEX` de la station mobile (RNX_header_mobile, RNX_data_mobile)
- les données de navigation (NAV_header, NAV_data, qui peuvent aussi être remplacées par $sp3\_header$ et $sp3\_data$ si des orbites précises sont utilisées)
- l'époque à laquelle est réalisée le calcul
- les coordonnées de la station de base $X0\_base$
- les coordonnées approchées de la station mobile $X0\_mobile$

Le vecteur sat_index est remplacé par les vecteurs sat_const et sat_PRN, qui contiennent respectivement le numéro de la constellation de chaque satellite visible (1 : GPS, 2 : Galileo, 3 : Glonass), ainsi que leur PRN.

Récupération des observations communes aux deux récepteurs

Afin d'appliquer les corrections PRC, les observations des stations de base et mobile doivent être réalisées au même instant. Pour une époque donnée du `RINEX` de la station de base, il faut donc récupérer les données correspondantes dans le `RINEX` de la station mobile.

À programmer...
Dans la fonction `pretraitements_DGNSS()`, récupérer les observations communes pour une époque donnée. Récupérer ensuite l'éphéméride de chaque satellite visible.

On pourra s'aider de la fonction `get_obs_in_common()` de la `yagps_toolbox` pour obtenir les observations simultanées entre les deux stations.

Les données ainsi chargées pourront être sauvegardées dans deux tableaux cellulaires - un par station -, afin de faciliter leur accès dans la suite de ce TD. Les informations nécessaires

à sauvegarder pour chaque satellite visible sont :

- la constellation
- le PRN
- le mjd
- la pseudo-distance utilisée dans le calcul ($C1$, $C2$ ou bien la combinaison $ionosphere - free$)
- la structure Ephemeride correspondant à l'observation

Calcul des corrections et de la position des satellites pour la station de base

À programmer...

Calculer les pseudo-distances corrigées, ainsi que la position des satellites pour la station de base, en utilisant le code déjà développé dans la fonction `pretraitements_spp()`.

Calcul des corrections PRC

Les coordonnées de la station de base étant connues, il est possible de calculer les PRC à partir des résidus issus de la compensation avec la position contrainte.

À programmer...

Calculer le PRC de chaque satellite à l'aide de la fonction `trilat()`. On rappelle que $PRC^j = -V^j$ pour un satellite j.

Correction des pseudo-distances de la station mobile

Les pseudo-distances observées par la station mobile peuvent ensuite être corrigées.

À programmer...

Corriger les mesures de pseudo-distance observées par le récepteur mobile :

$$PR^j_{corr,mobile} = PR^j_{mobile} + PRC^j$$

Les tableaux sat_const et sat_PRN vont permettre de retrouver la bonne correction.

Calcul des corrections et de la position des satellites pour la station de base

Il reste alors à prendre en compte le reste des corrections (erreur d'horloge récepteur, troposphere, rotation de la terre, etc ...) pour les pseudo-distances de la station mobile, ainsi qu'à calculer la position des satellites visibles depuis la station mobile.

 À programmer...

Calculer les pseudo-distances corrigées, ainsi que la position des satellites pour la station mobile, en utilisant le code déjà développé dans `pretraitements_spp()`.

Prise en compte de la hauteur d'antenne

Avant de lancer un calcul DGNSS, il reste à prendre en compte la hauteur de l'antenne de la station de base. En effet, la position de la station donnée dans l'en-tête des fichiers `RINEX` d'observation ne prend pas en compte la hauteur d'antenne. Cet écart - entre la position officielle de la station et l'ARP de l'antenne GNSS - peut atteindre 1 voire 2 mètres, ce qui va "fausser" la correction PRC si elle n'est pas prise en compte.

 À programmer...

Transformer les coordonnées de la station de base en coordonnées de l'ARP de l'antenne GNSS.

Les offsets d'antennes, présents dans RNX_header, sont exprimés dans un repère local, et correspondent ainsi aux coordonnées de l'ARP dans un repère local centré sur la position officielle du point. Il suffit alors de retransformer ces coordonnées locales en coordonnées cartésiennes, en gardant pour origine du repère local les coordonnées officielles.

On pourra utiliser la fonction de transformation de coordonnées `tool_loccart_GRS80()`.

 `pygpstoolbox` ...

Cette correction est effectuée dans la fonction

```
[X, Y, Z] = corr_pos_atx(X, Y, Z, RNX_header,RNX_2_pos)
```

Calcul de la position du recepteur mobile

Il est maintenant possible d'effectuer un positionnement classique sur le code avec les données corrigées.

À programmer...
Écrire un script `run_calcul_DGNSS()` qui calcule la position de la station $mlvl$ avec pour station de base $smne$. Cette fonction appellera les foctions `pretraitements_DGNSS()` et `trilat()`.

Les coordonnées de la station $mlvl$ estimées pourront ensuite être corrigées de la hauteur d'antenne afin de pouvoir être comparées aux coordonnées officielles de la station (voir TD 11).

Ajout des constellations Glonass et Galileo

Si les parties 8 et 12 du TP ont été réalisées, il est facile de prendre en compte les constellations Glonass et Galileo. En effet les corrections PRC sont calculées de la même manière. Seuls les offsets $GGTO$ et $GPGL$ doivent être estimés en plus lors du calcul des PRC, ainsi que lors du calcul de la position de la station mobile.

Analyse des résultats

Comparer les résultats obtenus par DGNSS à ceux obtenues par positionnement spp standard.

Comparer les résultats obtenus par un calcul avec C1 ou la combinaison $ionosphere-free$. On observera que la précision du positionnement sera meilleure avec C1. En effet, pour un calcul DGNSS, il subsiste pricipalement l'erreur de mesure de code qui augmente considérablement avec la combinaison $ionosphere - free$.

ENSG Géomatique — ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES	Calcul de la position d'un récepteur GNSS "à la main"	uTOP Université de Technologie Ouverte Pluripartenaire
Partie : 15	Calcul d'une position à partir des mesures de phase par doubles différences	Difficulté : ⋆⋆⋆

Principe du positionnent par doubles différences

Le positionnement sur la phase par doubles différences est une technique de positionnement différentielle qui permet d'éliminer les erreurs d'horloge satellite et récepteur, ainsi que de limiter l'impact des erreurs d'orbite et du retard troposphérique et ionosphérique, liés à la propagation des ondes électromagnétiques dans l'atmosphère.

Il sera ainsi nécessaire de disposer de deux stations qui observent simultanément, avec une station dont les coordonnées sont connues (station de base).

Le bruit de mesure de la phase (2 mm) étant bien inférieur au bruit de mesure du code (1 m), le positionnement sur la phase sera bien plus précis que le positionnement sur le code. Cependant, ce type de positionnement nécessite la détection, ainsi que la résolution des ambiguités.

Cette technique de correction est d'autant plus efficace que la distance entre les deux stations est faible. Elle est efficace jusqu'à quelques dizaines de kilomètres entre les récepteurs.

La précision ainsi atteinte est millimétrique à centimétrique en fonction des conditions d'observation et de la longueur de la ligne de base.

Mise en équations

La formulation simplifiée de la mesure de phase entre un satellite j et un récepteur i est :

$$l_i^j = \rho_i^j + c \times dt_{r,i} - c \times dt_e^j + d_{tropo,i}^j - \lambda \times N_i^j$$

où

- l_i^j : mesure de phase, en distance
- $\rho_i^j = \sqrt{(X_i - X^j)^2 + (Y_i - Y^j)^2 + (Z_i - Z^j)^2}$: distance géométrique entre le récepteur i et un satellite j
- $dt_{r,i}$: erreur d'horloge du récepteur i
- dt_e^j : erreur d'horloge du satellite j

— $d^j_{tropo,i}$: retard troposphérique
— N^j_i : ambiguité de la mesure
— λ : longueur d'onde du signal utilisé
— c : célérité de la lumière dans le vide $c = 299792458.0\,\mathrm{m.s^{-1}}$

Le terme $d^j_{iono,i}$ est volontairement omis, nous considérerons par la suite que le calcul est réalisé en utilisant la combinaison *ionosphere-free* afin d'éliminer l'erreur liée au délai ionosphérique.

De plus les récepteurs mesurent le nombre de cycles $\Delta\varphi^j_i$. On a alors $l^j_i = \lambda \times \Delta\varphi^j_i$, avec λ : longueur d'onde du signal (ici L_3).

Considérons un autre récepteur k, observant le satellite j à la même époque que le récepteur i, et calculons la différence des phases mesurées par i et k (simple différence) :

$$l^j_{i,k} = l^j_k - l^j_i = \rho^j_k - \rho^j_i + c(dt_{r,k} - dt_{r,i}) + d^j_{tropo,k} - d^j_{tropo,i} - \lambda(N^j_k - N^j_i))$$

Le terme d'horloge dt^j_e est éliminé et les effets des erreurs d'orbites et de propagation de l'onde dans l'atmosphère sont réduits.

Considérons alors un autre satellite, l, visible par les deux récepteurs i et k à la même époque. Calculons la différence des simples différences sur ces deux satellites :

$$l^{j,l}_{i,k} = l^l_{i,k} - l^j_{i,k} = \rho^{j,l}_{i,k} + d^{j,l}_{tropo,i,k} - \lambda N^{j,l}_{i,k}$$

avec :

$$\begin{cases} l^{j,l}_{i,k} &= l^l_k - l^l_i - l^j_k + l^j_i \\ \rho^{j,l}_{i,k} &= \rho^l_k - \rho^l_i - \rho^j_k + \rho^j_i \\ \tau^{j,l}_{i,k} &= d^l_{tropo,k} - d^l_{tropo,i} - d^j_{tropo,k} + d^j_{tropo,i} \\ N^{j,l}_{i,k} &= N^l_k - N^l_i - N^j_k + N^j_i \end{cases}$$

Cette formulation est appelée doubles différences, et élimine l'erreur d'horloge des récepteurs i et k.

Calcul de la position par doubles différences

Dans un premier temps on ne s'intéressera qu'à la constellation GPS.

Pour calculer la position du récepteur mobile par doubles différences, les étapes suivantes devront être réalisées :

1. Préparation des données : récupération des observations simultannées sur les deux récepteurs, pour une époque donnée, calcul de la combinaison $ionosphere - free$ et récupération des éphémérides correspondantes aux observations. Calcul qui sera réalisé dans `pretraitements_phase()`.

2. Estimation de l'erreur d'horloge récepteur pour les deux stations, afin de connaître

précisément la position des satellites aux instants d'émission. Calcul de la position des satellites et des corrections du délai troposphérique. Calcul qui sera réalisé dans `pretraitements_phase()`.

3. Détection des discontinuités dans l'observation des satellites. Calcul qui sera réalisé dans `detect_discontinuites()`.

4. Sélection des satellites pivots. Calcul qui sera réalisé dans `selection_pivots()`.

5. Formation des doubles différences et résolution du système par moindres carrés. Calcul qui sera réalisé dans `MC_phase()`.

Par la suite nous pourrons améliorer la qualité du positionnement en ajoutant :

— une détection des sauts de cycle

— la prise en compte de la hauteur d'antenne

— la prise en compte du modèle d'antenne

— les constellations Glonass et Galileo

Fonction `pretraitements_phase()`

La fonction `pretraitements_phase()` va remplacer la fonction `pretraitements_spp()`.

À programmer...
Écrire la fonction :

```
[t, Pos_sat, Dobs, PobsL1, PobsL2, PobsL3, Dtropo, ElevSat, sat_const,
sat_PRN] = pretraitements_phase( RNX_header_base, RNX_data_base,
RNX_header_mobile, RNX_data_mobile, NAV_header, NAV_data ,epoque,
X0_base, X0_mobile )
```

Cette fonction doit prendre en entrée :

— les observations `RINEX` de la station de base (RNX_header_base, RNX_data_base)

— les observations `RINEX` de la station mobile (RNX_header_mobile, RNX_data_mobile)

— les données de navigation (NAV_header, NAV_data, qui peuvent aussi être remplacées par $sp3\_header$ et $sp3\_data$ si des orbites précises sont utilisées)

— l'époque à laquelle est réalisée le calcul

— les coordonnées de la station de base $X0\_base$

— les coordonnées approchées de la station mobile $X0\_mobile$

Elle calcule en sortie :

— t : mjd des observations (une ligne par satellite)

— $PosSat$: position des satellites [X, Y, Z] en mètres (une ligne par satellite)

— $Dobs$: pseudo-distances corrigées en mètres (une ligne par satellite, une colonne par récepteur)

— $PobsL1$, $PobsL2$, $PobsL3$: mesures de phase en mètres, sur les deux fréquences, et combinaison $ionosphere - free$ (une ligne par satellite, une colonne par récepteur)

— $Dtropo$: correction du délai troposphérique en mètres (une ligne par satellite, une colonne par récepteur)

— $ElevSat$: élévation des satellites en radian (une ligne par satellite, une colonne par récepteur)

— sat_const : indice de la constellation des satellites (1 : GPS, 2 : Galileo, 3 : Glonass) (une ligne par satellite)

— sat_PRN : PRN des satellites (une ligne par satellite)

La première colonne des différentes matrices correspondra à la station de base et la seconde à la station mobile.

`pygpstoolbox` ...

La fonction `pretraitements_phase()` peut être remplacée par la fonction :

```
interf_calc_preprocessing_phase()
```

de la `yagps_toolbox`.

Préparation des données

Dans un premier temps, il faut récupérer, pour chaque époque les données communes aux stations de base et mobile. Pour une époque donnée du `RINEX` de la station de base, il faut donc récupérer les données correspondantes dans le `RINEX` de la station mobile.

À programmer...

Dans la fonction `pretraitements_phase()`, récupérer les observations communes pour une époque donnée. Récupérer ensuite l'éphéméride de chaque satellite visible. Enfin, calculer la combinaison $ionosphere - free$ sur le code et la phase.

On pourra s'aider de la fonction `get_obs_in_common()` de la `yagps_toolbox` pour obtenir les observations simultanées entre les deux stations.

Les données ainsi chargées pourront être sauvegardées dans deux tableaux cellulaires - un par station -, afin de faciliter leur accès dans la suite de ce TD. Les informations nécessaires à sauvegarder pour chaque satellite visible sont :

- la constellation
- le PRN
- le mjd
- la pseudo-distance $ionosphere - free$
- les observations de phase (L1, L2 et L3) transformées en distances
- la structure Ephemeride correspondant à l'observation

Calcul de la position des satellites aux instants d'émission, calcul de la correction du délai troposphérique

Depuis le début du TP, l'erreur d'horloge du récepteur dt_r n'est pas prise en compte dans le calcul de l'instant d'émission des signaux GNSS, mais seulement estimée. Les horloges du récepteur et du système GPS ne sont pas parfaitement synchronisées, avec un décalage qui peut atteindre l'ordre de grandeur de la milliseconde. Les satellites GNSS ayant une vitesse d'environ 4 km/s, ce décalage peut engendrer une erreur de plusieurs mètres sur le calcul de la position des satellites.

L'idée est donc d'estimer une première fois l'erreur d'horloge récepteur dt_r, et de calculer la position des satellites en prenant en compte cette erreur.

À programmer...

Calculer les pseudo-distances corrigées, ainsi que la position des satellites pour les deux stations, en se basant sur le code développé dans la fonction `pretraitements_spp()`.

Estimer ensuite l'erreur d'horloge du récepteur de la station de base à l'aide de la fonction `trilat()`. On pourra contraindre la position de la station de base (Voir le TD 14 pour ajouter les contraintes).

Corriger alors les pseudo-distances de l'erreur d'horloge récepteur :

$$PR_{corr} = PR + c \times dt_r$$

Calculer ensuite les pseudo-distances corrigées, ainsi que la position des satellites pour la station de base, aux instants d'émission des signaux GNSS, en se basant sur le code développé dans la fonction `pretraitements_spp()`.

Il doit y avoir le même nombre d'observations entre les deux récepteurs. Si un satellite n'est pas observé depuis un récepteur (par exemple, si son élévation est trop faible), supprimer les observations correspondantes de la seconde station.

A ce stade, sont calculées les positions des satellites, leur élévation, ainsi que la correction du modèle troposphérique. Remplir les matrices $PosSat$, $PobsL1$, $PobsL2$, $PobsL3$, $Dtropo$, $ElevSat$, sat_const et sat_PRN.

Dans un script `run_calcul_phase()`, effectuer ce calcul pour chaque époque, entre les station $smne$ (base) et $mlvl$ (mobile).

Concaténer ensuite les matrices obtenues et créer un vecteur t contenant la date (au format mjd) de chaque observation.

Détection des discontinuités

Les ambiguités N n'ont pas à être résolues pour chaque observation. Tant qu'il n'y a pas d'interruption du signal, c'est à dire que le satellite reste "accroché" au récepteur, la valeur de l'ambiguité à estimer reste identique pour un satellite donné.

Afin de déterminer quelles ambiguités sont à estimer, il faut pour chaque satellite :

- déterminer les plages d'observations contigües, pour lesquelles il n'y a pas d'interruptions du signal. Toute interruption entraîne l'estimation d'une nouvelle ambiguité.
- détecter les sauts de cycles : une nouvelle ambiguité peut apparaître sans que l'interruption du signal ne soit visible. Les sauts de cycles peuvent être causés par des obstructions (arbres, bâtiments, etc...)

Nous allons dans un premier temps uniquement déterminer les plages d'observations contigües, quelles ambiguités doivent être estimées et calculer la valeur initiale de ces ambiguités.

À programmer...

Écrire une fonction `detect_discontinuites()`, qui détecte pour les observations des station de base et mobile, les discontinuités dans l'observation des signaux GNSS.

Le mjd des époques observées peut être obtenu par :

```
mjd_epoques = unique(t);
```

```
2
```

Une discontinuité sera alors détectée lorsque deux observations successives pour un satellite ne correspondront pas à deux époques successives du calcul.

Une nouvelle ambiguité sera ainsi introduite :

— à la première observation du satellite

— après chaque discontinuité

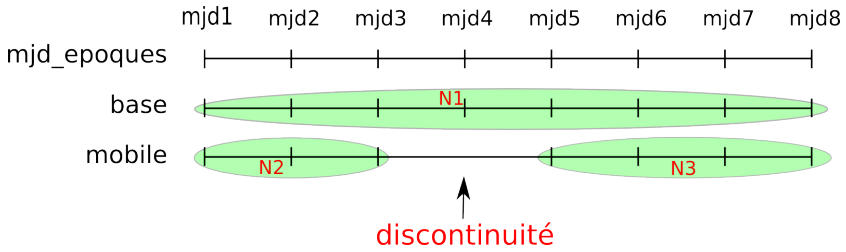


FIG 9 - Détection des discontinuités, et affectation des ambiguités (N1 à N3). L'observation à la date mjd4 devra être supprimée de la station de base afin de conserver le même nombre d'observations entre récepteurs

Une valeur approchée des ambiguités pourra être calculée par $amb0 = P3 - L3$.

Attention à disposer de suffisamment d'observations pour estimer une ambiguité (minimum 15/20 observations). Dans le cas contraire, supprimer les observations correspondant à cette ambiguité. Conserver le même nombre d'observations entre les deux récepteurs.

Tester la fonction dans `run_calcul_phase()`.

`pygpstoolbox` ...

Cette fonction peut être remplacée par la fonction :

```
1 [t, Pobs, Dobs, PosSat, ElevSat, AzSat, Dtropo, sat_index,
2 amb_index, amb0] = calc_cycle_slip(t, L1, L2, Pobs, Dobs,
3 PosSat, ElevSat, AzSat,Dtropo, sat_index)
```

de la `yagps_toolbox`.

Sélection des satellites pivots

Dans la formation des doubles différences, deux satellites sont nécessaires. Un des satellites est appelé satellite pivot : ce sera par rapport à ce satellite que les doubles différences seront construites. L'idée est donc de sélectionner un satellite pour lequel les données sont le moins entachées d'erreurs, c'est à dire un satellite dont l'élévation est maximale pour minimiser les erreurs liées à la propogation de l'onde dans l'atmosphère.

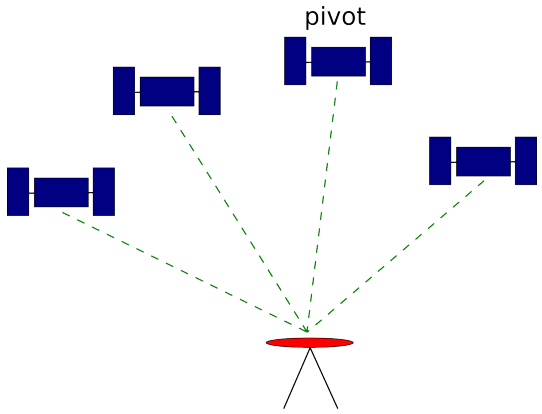


FIG 10 - Sélection du satellite pivot

À programmer...

Écrire une fonction `selection_pivots()`, dans laquelle sont sélectionnés les satellites pivots. On pourra choisir un satellite pivot pour chaque heure d'observation afin de toujours sélectionner le satellite le plus haut dans le ciel. On veillera à ce que le satellite soit présent durant tout l'intervalle ($\pm$ 2 époques).

Tester la fonction dans `run_calcul_phase()`.

`pygpstoolbox` ...

Cette fonction peut être remplacée par la fonction :

```
[pivots, interv] = calc_pivot(t, ElevSat, sat_index)
```

de la `yagps_toolbox`.

Formation des doubles différences et calcul de la position du récepteur mobile

L'équation des doubles différences est la suivante :

$$l_{i,k}^{j,l} = \rho_{i,k}^{j,l} + d_{tropo,i,k}^{j,l} - \lambda N_{i,k}^{j,l}$$

avec :

$$\begin{cases} l_{i,k}^{j,l} &= l_k^l - l_i^l - l_k^j + l_i^j \\ \rho_{i,k}^{j,l} &= \rho_k^l - \rho_i^l - \rho_k^j + \rho_i^j \\ \tau_{i,k}^{j,l} &= d_{tropo,k}^l - d_{tropo,i}^l - d_{tropo,k}^j + d_{tropo,i}^j \\ N_{i,k}^{j,l} &= N_k^l - N_i^l - N_k^j + N_i^j \end{cases}$$

Avec la convention suivante :

- i : station de base
- k : station mobile
- j : satellite pivot
- l : autre satellite

cette équation peut être réorganisée, entres parties fixes, à gauche du signe égal, et parties variables, à droite :

$$l_{i,k}^{j,l} - (\rho_i^j - \rho_i^l) - d_{tropo,i,k}^{j,l} = \rho_k^l - \rho_k^j - \lambda N_{i,k}^{j,l}$$

Sont ainsi à estimer :

- la position du récepteur mobile X, Y, Z.
- les doubles différences d'ambiguités $N_{i,k}^{j,l}$

À programmer...

Le calcul sera effectué dans la fonction :

```
[X, Y, Z, DDN, V, Vnorm, sigma02, Qxx] = MC_phase(t, Pobs, PosSat,
ElevSat, Dtropo, sat_const, sat_PRN, index_amb, amb0, pivots, interv,
X0_base, X0_mobile)
```

En entrée de la fonction :

- Date de chaque observation t (format mjd), sous la forme d'un vecteur (j x 1)
- Observations de phase $Pobs$, sous forme d'une matrice (j x 2). Une colonne par récepteur
- Coordonnées des satellites $PosSat$ sous forme d'une matrice (j x 3)
- Elevation des satellites $ElevSat$, sous forme d'une matrice (j x 2). Une colonne par récepteur

— Correction du delai tropospherique $Dtropo$, sous forme d'une matrice (j x 2). Une colonne par récepteur

— Identifiants des satelllites sat_const (pour la constellation) et sat_PRN (pour le PRN), sous la forme de deux vecteurs (j x 1)

— Indice des ambiguités $index\_amb$, sous forme d'une matrice (j x 2). Une colonne par récepteur

— Valeur approchée des ambiguités $amb0$, sous la forme d'un vecteur (j x 1)

— L'indice des satellites pivots $pivots$, sous forme d'une matrice (j x 2) [const, PRN], ainsi que les intervalles $interv$ dans lesquels les satellites pivots sont valables, sous forme d'une matrice (j x 2) [mjd_min, mjd_max]

— Les coordonnées de la station de base $X0\_base$ [X ; Y ; Z], ainsi que les coordonnées approchées de la station mobile $X0\_mobile$ [X ; Y ; Z]

En sortie de la fonction :

— Position du récepteur (X, Y, Z)

— Doubles différences sur les ambiguités estimées : DDN

— Vecteur V des résidus

— Vecteur $Vnorm$ des résidus normalisés

— Estimateur du facteur unitaire de variance $\hat{\sigma}_0^2 = \frac{V^t PV}{n-p}$

— Matrice de variance-covariance sur les paramètres Σ_X (Qxx)

La première étape consiste à calculer la partie fixe de l'équation des doubles différences. Pour chaque époque, les doubles différences seront formées entre les satellites visibles et le satellite pivot correspondant.

Il faudra aussi déterminer quelles doubles différences $N_{i,k}^{j,l}$ sont à estimer, ainsi que calculer leur valeur initiale à partir de $index\_amb$ et $amb0$. Un nouveau $N_{i,k}^{j,l}$ sera à estimer, lorsque pour le même couple de satellites j, l, une nouvelle ambiguité interviendra.

Exemple : Epoque 1, couple j, l :

$$PosSat(ep\ 1, sat\ j\ et\ l) = \begin{pmatrix} X_j & Y_j & Z_j \\ X_l & Y_l & Z_l \end{pmatrix},\ index\_amb(ep\ 1, sat\ j\ et\ l) = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

$\rightarrow N_{i,k}^{j,l}$ formé à partir des ambiguités 1, 2, 3 et 4.

Epoque n, couple j, l, ambiguité différente pour le satellite j observé par la station i

$$PosSat(ep\ n, sat\ j\ et\ l) = \begin{pmatrix} X_j & Y_j & Z_j \\ X_l & Y_l & Z_l \end{pmatrix},\ index\_amb(ep\ n, sat\ j\ et\ l) = \begin{pmatrix} 5 & 3 \\ 2 & 4 \end{pmatrix}$$

$\rightarrow N_{i,k}^{j,l}$ formé à partir des ambiguités 5, 2, 3 et 4.

Il y aura ainsi deux $N_{i,k}^{j,l}$ différents à estimer pour le couple de satellites j, l.

La dernière étape consiste alors à former les matrices A et B des moindres carrés, et à estimer les différentes variables.

La pondération du système pourra être réalisée en posant :

$$\sigma_{DD}^2 = \sigma_{i,j}^2 + \sigma_{i,l}^2 + \sigma_{k,j}^2 + \sigma_{k,l}^2$$

avec :

$$\sigma_{i,j} = \frac{\sigma_{phase}}{\sin Z_i^j}$$

où σ_{phase} = 2 cm et Z_i^j est la distance zénithale entre le récepteur i et le satellite j.

 `pygpstoolbox` ...

Cette fonction peut être remplacée par la fonction :

```
1 interf_calc_LS_phase()
```

de la `yagps_toolbox`.

Calcul

 À programmer...

Dans la fonction `run_calcul_phase()`, calculer la position de la station $mlvl$ à partir de la station $smne$ avec la constellation GPS.

Pour aller plus loin

Prise en compte de la hauteur et du modèle d'antenne

Les coordonnées de la station de base doivent correspondre aux coordonnées du centre de phase du récepteur, ce qui n'est pas le cas pour l'instant. Deux vecteurs sont à prendre en compte : le vecteur allant de la position "officielle" de la station à l'ARP de l'antenne ("hauteur d'antenne"), ainsi que le vecteur allant de l'ARP au centre de phase de l'antenne (calibration d'antenne). La partie variable de ce dernier vecteur ne sera pas prise en compte.

De même, les coordonnées de la station mobile sont calculées au niveau du centre de phase de l'antenne. Il faut aussi tenir compte de ces vecteur pour comparer ses coordonnees aux coordonnées officielles.

À programmer...

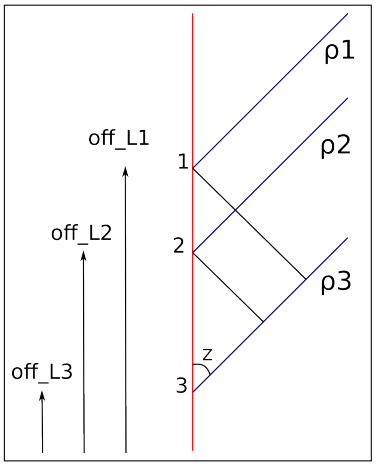
La hauteur d'antenne est donnée dans l'en-tête du fichier RINEX (champs dE, dN et dH). La calibration de l'antenne peut être obtenue avec le code suivant (ne prendre en compte que la composante UP) :

```
% Un fichier igs08.atx est deja present dans la toolbox
[ATX_header, ATX_data] = load_antex('igs08.atx');
ATX_name = RNX_header.ANT_TYPE;
[ATX_f1] = get_antex(ATX_header, ATX_data, ATX_name,'G01' );
[ATX_f2] = get_antex(ATX_header, ATX_data, ATX_name,'G02' );

% correction sur f3
dU_calib = 2.5*ATX_f1.UP-1.5*ATX_f2.UP;
```

Les corrections sont données dans un rempère local. On pourra utilser la fonction `tool_loccart_GRS80()` pour appliquer les corrections.

Démonstration : correction de la calibration d'antenne sur L3 en UP



Nous avons :

$$\begin{cases} PR_1 &= \rho_1 + \cdots \\ PR_2 &= \rho_2 + \cdots \\ PR_3 &= \rho_3 + \cdots \end{cases}$$

Or :

$$\rho_3 = \frac{\alpha^2}{\alpha^2 - \beta^2} \cdot \rho_1 - \frac{\beta^2}{\alpha^2 - \beta^2} \cdot \rho_2$$

Avec :

$$\begin{aligned} \rho_3 &= \rho_1 + d_{13} \cdot \cos z \\ \rho_2 &= \rho_1 + d_{12} \cdot \cos z \end{aligned}$$

D'où :

$$\begin{aligned}
\rho_1 + d_{13} \cdot \cos z &= \tfrac{\alpha^2}{\alpha^2-\beta^2} \cdot \rho_1 - \tfrac{\beta^2}{\alpha^2-\beta^2} \cdot (\rho_1 + d_{12} \cdot \cos z) \\
d_{13} \cdot \cos z &= -\tfrac{\beta^2}{\alpha^2-\beta^2} \cdot d_{12} \cdot \cos z \\
d_{13} &= -\tfrac{\beta^2}{\alpha^2-\beta^2} \cdot d_{12} \\
&\approx -1,5 \cdot d_{12}
\end{aligned}$$

Ce qui implique que :

$$\begin{aligned}
off_{L3} &= off_{L1} - d_{13} \\
&= off_{L1} + 1,5 \cdot d_{12} \\
&= off_{L1} + 1,5 \cdot (off_{L1} - off_{L2}) \\
&= 2,5 \cdot off_{L1} - 1,5 \cdot off_{L2}
\end{aligned}$$

pygpstoolbox ...

Cette correction est effectuée dans la fonction

```
[X, Y, Z] = corr_pos_atx(X, Y, Z, RNX_header,RNX_2_pos)
```

Détection des sauts de cycle

Plusieurs méthodes permettent la détection et l'étude des sauts de cycles :

- la formation de triples différences sur les mesures de phase, qui a pour avantage d'éliminer les ambiguités entières inconnues
- la formation de la combinaison *geometry-free*, $L4 = L1 - L2$, qui permet le calcul des valeurs approchées des ambiguités. Un saut de cycle est alors détecté lorsque $\frac{d^2L4}{dt^2} >$ seuil, et que deux pics d'amplitudes égales et opposés se suivent

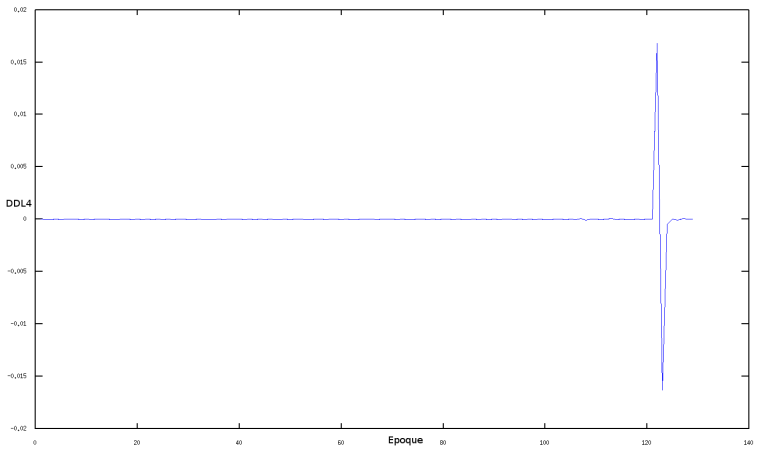


FIG 11 - Exemple de saut de cycle détecté

À programmer...

Dans la fonction `detect_discontinuites()`, ajouter une détection des sauts de cycle en utilisant la seconde méthode.

Commencer par former $\frac{d^2L4}{dt^2}$ en utilisant la formule de dérivation discrète :

$$\frac{d^2L4}{dt^2}(i) = \frac{L4(i+1) - 2 \times L4(i) + L4(i-1)}{(t(i+1) - t(i-1))^2}$$

Attention à passer t en secondes.

Détecter les sauts de cycle. On utilisera un seuil de 0.0001. Chaque saut de cycle entraîne

l'apparition d'une nouvelle ambiguité.

Attention à disposer de suffisamment d'observations pour estimer une ambiguité (minimum 15/20 observations). Dans le cas contraire, supprimer les observations correspondant à cette ambiguité. Conserver le même nombre d'observations entre les deux récepteurs.

 pygpstoolbox ...

Cette fonction peut être remplacée par la fonction :

```
[t, Pobs, Dobs, PosSat, ElevSat, AzSat, Dtropo, sat_index,
amb_index, amb0] = calc_cycle_slip(t, L1, L2, Pobs, Dobs,
PosSat, ElevSat, AzSat, Dtropo, sat_index)
```

de la yagps_toolbox.

Ajout des constellations Galileo et Glonass

Les constellations Galileo et Glonass peuvent facilement être ajoutées au calcul. En effet, les offsets, $GGTO$ et $GPGL$ sont éliminés dès la construction des simples différences.

On peut alors faire l'approximation que $\lambda_{3,Glonass} \approx \lambda_{3,Galileo} \approx \lambda_{3,GPS}$, et donc former les doubles différences d'ambiguités de le même manière qu'avec GPS seul, sans se soucier de la valeur de λ. Avec cette approximation, il ne sera pas possible de fixer les ambiguités, mais elle sera sans influence sur la précision du calcul de la position de la station mobile.

 À programmer...

Prendre en compte les constellations Glonass et Galileo en plus de GPS dans le calcul.

Les trois méthodes de positionnement présentées dans le TD ont été réalisées dans la bibliothèque `yagps_toolbox`. Leur syntaxe est présentée ci-dessous :

Positionnement spp :

```
[result,interm_results] = run_spp(rinex_o,rinex_n,options)
```

Positionnement DGNSS :

```
[result_base,interm_results_base,result_rover,
   interm_results_rover] =
run_DGNSS(rinex_o_base,rinex_o_rover,rinex_n,X0_base,options)
```

Positionnement différentiel sur la phase :

```
[result,interm_results,result_LS_code_base,
   result_LS_code_rover] =
run_phase(rinex_o_base,rinex_o_rover,rinex_n,X0_base,options)
```

Le paramètre $options$ permet de définir les différentes options du calcul (constellation(s) utilisée(s), angle de coupure, répertoire de sauvegarde des graphiques, etc...). Voir la documentation utilisateur, ou tapper 'help nom_fonction' pour obtenir la syntaxe complète.

Etude du positionnement spp

Comparer les résultats d'un positionnement de la station $smne$ avec GPS seul, Glonass seul, Galileo seul, GPS + Galileo et GPS + Glonass + Galileo. On pourra se limiter à 200 époques.

Comparer ensuite les calculs réalisés sur $C1$, $C1$ avec modèle ionosphérique de Klobuchar et sur la combinaison $ionosphere - free$.

On pourra comparer les résultats avec la fonction `plot_plani()` qui permet la représentation des positions calculées en 2 dimensions dans un repère local. Les coordonnées locales sont

stockés dans les champs E et N du tableau de structure $result$:

```
E = [];
N = [];
for i = 1:length(result)
    E = [E;result(i).E];
    N = [N;result(i).N];
end
```

On pourra aussi visualiser les différentes séries temporelles à l'aide de la fonction `plot_graph()`.

Etude du positionnement DGNSS

Réaliser les même comparaisons pour le positionnement DGNSS, entre les station $mlvl$ (base) et $smne$ (mobile). Les coordonnées sont strockées dans le tableau de structures $result\_rover$.

Comparaison spp/DGNSS

Comparer les résultats obtenus par un positionement spp et un positionnement DGNSS.

Etude du positionnement sur la phase

Réaliser des calculs sur la phase avec différentes constellations, et sur des périodes plus ou moins longues.

Pour aller plus loin...

Etudier l'influence de la longueur de la ligne de base, en DGNSS et pour un calcul sur la phase par doubles différences.

Annexe : liste des TD et parcours possibles

Liste des TD

Le TD 'GNSS à la main' se compose de 16 TD différents, qui ont pour objectif le calcul de positions GNSS. Les TD développés sont les suivants :

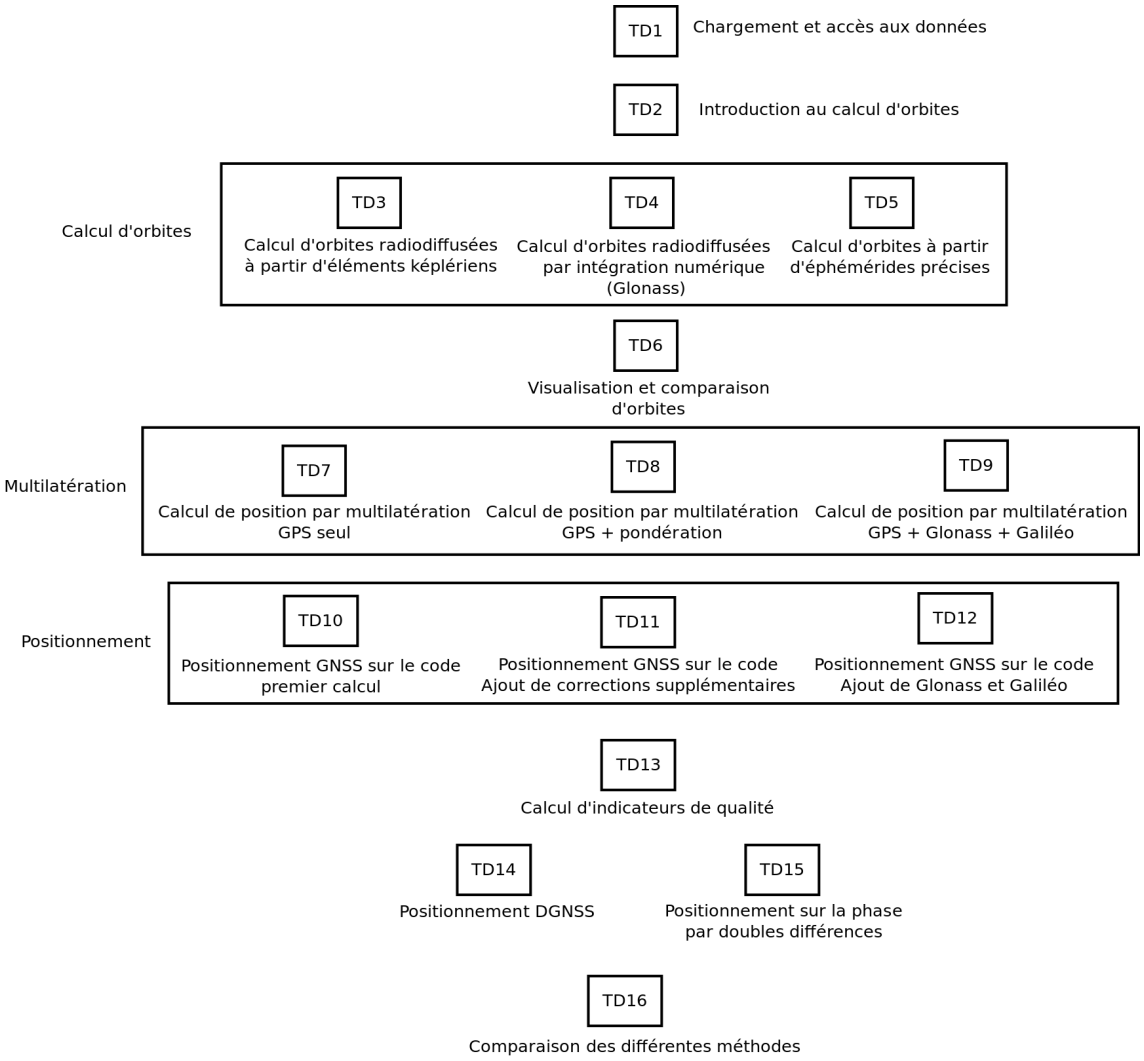


FIG 12 - TD

Exemple de parcours possibles

Il est possible de n'effectuer qu'une partie du TD, par exemple pour n'aborder que le positionnement sur le code. Différents 'parcours' sont présentés ci-dessous. Ils présentent l'enchainement des TD à réaliser pour atteindre différents objectifs :

1 - Positionnement absolu sur le code : TD 'minimal' pour calculer une position par GPS :

A réaliser, TD : 1 - 2 - 3 - 7 - 10 - 13

2 - Positionnement absolu sur le code : Ajout de corrections supplémentaires

A réaliser, TD : 1 - 2 - 3 - 7 - 8 - 10 - 11 - 13

3 - Positionnement absolu sur le code : Ajout des constellations Glonass et Galileo

A réaliser, TD : 1 - 2 - 3 - 4 - 5 - 7 - 8 - 9 - 10 - 11 - 12 - 13

4 - Positionnement différentiel sur le code (DGNSS) : GPS seul

A réaliser, TD : 1 - 2 - 3 - 7 - 8 - 10 - 11 - 13 - 14

5 - Positionnement différentiel sur le code (DGNSS) : GPS + Glonass + Galileo

A réaliser, TD : 1 - 2 - 3 - 4 - 5 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 14

6 - Positionnement différentiel sur la phase : GPS seul

A réaliser, TD : 1 - 2 - 3 - 7 - 8 - 10 - 11 - 13 - 15

7 - Positionnement différentiel sur la phase : GPS + Glonass + Galileo

A réaliser, TD : 1 - 2 - 3 - 4 - 5 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 15

Les TD 6 et 16 sont optionnels et ne nécessitent pas de réaliser les différents TDs. Le TD 6 permet de visualiser et de comparer différentes orbites et le TD 16 permet de comparer les différentes méthodes de positionnement présentées dans ce TD.