# DÉTERMINATION DE L'ORBITE D'UN ASTÉROÏDE

## ou d'une comète périodique

Sont rassemblés ici les éléments de compréhension et d'explication :
- des formulaires et algorithmes,
- du code des programmes effectuant les calculs .

Note commencée par Alain Leraut, le 21 décembre 2018

# Plan

# 1. BIBLIOGRAPHIE

1.1. Jean MEEUS, Calculs astronomiques à l'usage des amateurs, publié par la SAF, 1986

1.2. Jean MEEUS, Astronomical Algorithms, publié par Willmann-Bell Inc, 1991

1.3. Formulaire de calcul d'une éphéméride de petite planète ou de comète, à partir de ses éléments osculateurs, par Bruno Morando et Jean Chapron. Document au format PDF numérisé depuis une publication tapée à la machine (ou ressemblante).
Le document comprend une table des coordonnées rectangulaires du Soleil pour 1950 à 2000.
https://www.imcce.fr/content/medias/publications/publications-recherche/nst/docs/S003.pdf

1.4. Détermination d'une orbite parabolique à partir de trois observations, par Bruno Morando. Accessible depuis un site de l'université d'Harvard :
http://adsbit.harvard.edu/cgi-bin/nph-iarticle_query?bibcode=1989O%26T....20....3M&db_key=AST&page_ind=0&data_type=GIF&type=SCREEN_VIEW&classic=YES

# 2. EXÉCUTION DU PROGRAMME

## 2.1. Besoins

S'exécute en mode commande : il faut donc ouvrir un terminal.
Ne s'exécute qu'avec Python3.
Suppose la présence d'un fichier texte contenant les données de l'astre dans le même dossier que le code source.
Il est commode de « se déplacer vers ce dossier avant de lancer l'exécution ».
Exemple :

```
$ cd orbit/OrbitalDetermination35-master/
```

## 2.2. Exécution

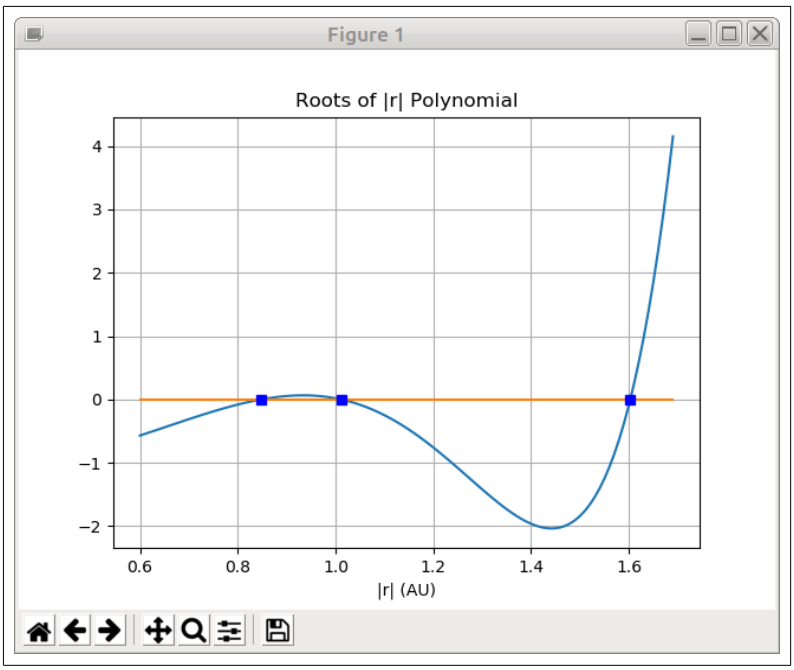
Le programme a été légèrement modifié en tenant compte de la remarque **3.2.** (page 6).

```
$ python3 main nom_du_fichier-de-données
```

*En utilisant le fichier de données joint, la commande devient :*

```
$ python3 main 1994PNinput.txt
```

Une fenêtre comportant une courbe s'ouvre alors :




Le programme fonctionne en exécutant des approximations de plus en plus précises (par itérations).

Ceci étant, il a besoin que l'utilisateur lui donne une « semence » pour situer le point de démarrage des itérations.

La partie de la courbe utilisable se situe ici entre 0,8 et 1,6 : on reste dans le « voisinage » de la ligne orange (quoiqu'elle puisse être).

Nous allons prendre 1,2 (soit 1.2 en notation Python).

*Fermer la fenêtre contenant le graphique*

Le programme affiche alors une demande d'introduction de donnée, par la question suivante :

```
Pick a root to try by index:
```

*Introduire alors 1.2 et validons par la touche Entrée*

Le programme affiche alors de nombreuses lignes qui traduisent les approximations successives.

Il finit par afficher les résultats qu'il a calculés et propose d'enregistrer ces résultats dans un fichier texte.

```
Orbital Elements (Epoch JD 2457597.125):
a: 2.31252903003
e: 0.534465825199
i: 45.341778295534084
O: 113.36740829641136
w: 234.6362575737111
Me: 338.0242957206046
------------------------------------------------------------------

Do you want to save results to file? (y/n):
```

## 2.3. Commentaires sur ces résultats

À rédiger

# 3. LE CODE : LES PRÉALABLES

## 3.1. Introduction

Analyse du code du programme « `main.py` » trouvé à l'adresse :
https://github.com/davenporta/OrbitalDetermination35

### 3.1.1. Présentation succinte.

Orbital Determination via the Method of Gauss
by Alexander Davenport
Original version 07/22/2016, SSP NMT 2016
Current version: 2.0.0 (09/19/2016)

## 3.2. Exécution du programme

Ne s'exécute qu'avec Python3.
Suppose la présence d'un fichier texte contenant les données de l'astre.
Exemple d'exécution :

```
$ cd orbit/OrbitalDetermination35-master/
$ python3 main.py 1994PNinput.txt
```

*Remarque :*

Pour limiter l'occupation du disque par des bibliothèques non nécessaires, éditer le code du programme et, ligne 4, désactiver l'appel à une bibliothèque KDE.

```
# matplotlib.use('Qt5Agg')
```

## 3.3. Le fichier de données utilisé pour les calculs

### 3.3.1. Contenu du fichier

| Ligne n° | |
|---|---|
| 1 | 10150 (1994 PN) |
| 2 | 2016-06-22 08:03:10.429 17:26:58.87 30:46:11.30 -0.0195340791840548 0.9323384266123208 0.4041408400938959 |
| 3 | 2016-06-30 07:19:24.379 17:08:12.58 27:19:02.51 -0.1537983334829756 0.9221125109705270 0.3997035045290656 |
| 4 | 2016-07-07 05:21:27.350 16:51:52.97 23:04:41.48 -0.2682470588825682 0.8998279354659114 0.3900449087482382 |
| 5 | 2016-07-13 03:42:48.670 16:38:37.64 18:31:04.40 -0.3635214279037087 0.8710218039698151 0.3775632429881836 |

**Résumé :**
5 lignes. La première contient le numéro de l'astéroïde : 10150 et sa désignation 1994 PN.
Chacune des suivantes contient, séparées par un espace, les informations suivantes (exemple de la ligne n°2) :

| *Contenu* | *Ce que c'est* |
|---|---|
| 2016-06-22 | La date de l'observation. Séparateur : tirets |
| 08:03:10.429 | L'heure de l'observation. Séparateur : deux points. |
| 17:26:58.87 | L'ascension droite (heure, minute et secondes d'arc) |
| 30:46:11.30 | La déclinaison (degrés, minutes, secondes d'arc) |
| -0.0195340791840548 | X |
| 0.9323384266123208 | Y |
| 0.4041408400938959 | Z |

Les valeurs X, Y, Z sont les coordonnées rectangulaires équatoriales géocentriques du Soleil.
L'origine de ces coordonnées est le centre de la Terre.
L'axe des X est dirigé vers le point vernal (longitude 0°).
L'axe des Y est contenu dans le plan de l'équateur et est dirigé vers la longitude 90°.
L'axe des Z est dirigé vers le pôle nord céleste.
(définitions d'après Jean Meeus, ref. 1.1 .chapitre 16). Voir également 1.2. chapitre 25).

Les valeurs X, Y, Z doivent être calculées à part en fonction du jour et de l'heure. Elles sont, ici, incorporées dans le fichier de données.
Si l'on envisage d'utiliser le programme sur un autre jeu de données (autre astéroïde ou comète, autres dates) il faudra calculer à part les valeurs X, Y, Z et les incorporer dans le fichier.
Ce calcul peut être effectué en utilisant les formulaires de Jean Meeus.

## 3.4. L'en-tête

Les lignes 1 à 7 du code initial

| *Code* | *Explications* |
|---|---|
| `import numpy as np` | |
| `from math import *` | |
| `import matplotlib` | |
| `# matplotlib.use('Qt5Agg')` | L'accès aux fonctionnalités spécifiques de l'environnement KDE est désactivé afin de limiter le « poids » des bibliothèques installées. |
| `import matplotlib.pyplot as plt` | |
| `import sys` | |
| `import re` | Module de traitement des expressions régulières |

## 3.5. Les « constantes »

Lignes 9 à 15 du code initial

| *Code* | *Signification* |
|---|---|
| `cAUD = 173.144632674` | |
| `timeEpoch = 2457597.125` | Numéro du jour julien servant de référence. |
| `k = 0.01720209895` | |
| `mu = 1.0` | |
| `ecA = radians(23.434)` | Obliquité de l'écliptique. C'est 23.4354 fin décembre 2018. |
| `asteroidDes = '1994PN'` | Valeur donnée par défaut. Elle sera actualisée en lignes 19 et 20 |

## 3.6. Lecture du fichier contenant les données observationnelles

Ligne 18

```
 data = np.loadtxt(str(sys.argv[1]), dtype='bytes',
skiprows=1).astype(str)
```

| Partie du code | Explications |
|---|---|
| `data` | Variable qui va contenir l'ensemble des données lues sauf la première ligne. |
| `(sys.argv[1])` | Le nom du fichier à lire est passé en argument dans la ligne de commande. |
| `str(sys.argv[1]),` | Convertit le résultat précédent en chaine de caractères |
| `dtype='bytes'` | Chacun des caractères lus est un octet |
| `skiprows=1` | La première ligne est sautée |
| `np.loadtxt(…).astype(str)` | Les données sont converties en mode string. On utilise la fonction np.loadtxt() pour lire un fichier de données, sans utiliser open ni close. |

*Expérimentation.*

```
 print(data) donne
 [['2016-06-22' '08:03:10.429' '17:26:58.87' '30:46:11.30'
   '-0.0195340791840548' '0.9323384266123208'
'0.4041408400938959']
  ...
  ['2016-07-13' '03:42:48.670' '16:38:37.64' '18:31:04.40'
   '-0.3635214279037087' '0.8710218039698151'
'0.3775632429881836']]
```

*Ligne 19.*

```
 with open(str(sys.argv[1]), 'r') as f:
     asteroidDes = re.search('\
((.*)\)',f.readline()).group(1).replace(" ","")
```

Lecture de la première ligne du fichier (qui contient `10150 (1994 PN)` ) et récupération de la seule désignation de l'astéroïde : `1994 PN`
(voir le contenu du fichier page 5).

## 3.7. Les « petites fonctions » de conversion

Conversions sexagésimal ↔ décimal, date → jour julien. Fonctions relativement simples à comprendre et faciles à tester avec un minimum de code.

### 3.7.1. De sexagésimal en décimal : lignes 23 à 35

Fonction appelée en lignes 62, 117, 118.

```
def sexToDecimal(ang, hparam):
```

| *Code* | *Explication* |
|---|---|
| ang | Peut être une ascension droite `08:03:10.429` ou une déclinaison `17:26:58.87` |
| `hparam` | Si ang est une ascension droite, le résultat est à multiplier par 15<br>`if hparam == True:`<br>`  ang *= 15` |
| `anglist = ang.split(":")` | Un contenu tel que `08:03:10.429` est transformé en [‘08’, ‘03’, ‘10.428’]<br>Cela permet ensuite le traitement par une boucle for<br>`for i in range(len(anglist)):`<br>`  anglist[i] = float(anglist[i])` |
| `decimal = (anglist[1] / 60) + (anglist[2] / 3600)` | Division des minutes par 60 et des secondes par 3600. (valeurs en float) |
| `if anglist[0] < 0:`<br>`  ang = anglist[0] - decimal`<br>`else:`<br>`  ang = anglist[0] + decimal` | Pas compris pour le moment |

*Exemple d'usage de la fonction.*

```
ra = '08:03:10.429'
AD = sexToDecimal(ra, True)
print("Ascension droite : {0} H:M:S ou {1} degres ".format(ra, AD))
deg = '17:26:58.87'
Dec = sexToDecimal(deg, False)
print("Declinaison      : {0} D:M:S ou {1} degres".format(deg, Dec))
```

*Donne :*

```
 Ascension droite : 08:03:10.429 H:M:S ou 120.79345416666668
degres
 Declinaison      : 17:26:58.87 D:M:S ou 17.44968611111111 degres
```

### 3.7.2. De décimal  vers sexagésimal : lignes 37 à 51

Fonction appelée en lignes (à préciser). Ne semble pas appelée.

### 3.7.3. Jour julien à partir du jour : lignes 54 à 59

Fonction appelée en ligne 116.

| *Code* | *Explication* |
|---|---|
| `def getJo(date):` | Paramètre : date contient la date du jour sous la forme d'une chaîne telle que '2016-06-22' |
| `return jodate` | Retourne un nombre au format float tel que 2457561.5 |

*Exemple d'usage :*

```
print(getJo('2016-06-22'))
```

*Résultat :*

```
2457561.5
```

### 3.7.4. Jour julien à partir du jour et de l'heure : lignes 61 à 63

Fonction appelée en ligne 116.

| *Code* | *Explication* |
|---|---|
| `def getJD(date, time):` | Paramètres :<br>- date contient la date du jour sous la forme d'une chaîne telle que '2016-06-22'<br>- heure sous la forme d'une chaîne telle que '08:03:10.429' |
| `return date + (UT / 24)` | Retourne un nombre au format float tel que 2457561.5 |

*Exemple d'usage :*

```
print(getJD(getJo('2016-06-22'), '08:03:10.429'))
```

*Résultat :*

```
2457561.8355373726
```

## 3.8. Fonction supposant plus de savoir mathématique

Le code est ici simple à écrire mais pas facile à comprendre pour qui ne comprend pas les matrices.

### Introduction de quelques notions

Pour comprendre ce qui se passe, il faut avoir une idée – non confuse – de ce qu'est une matrice et le calcul matriciel.

*Prenons un premier exemple :*

Calculer 7 * 6

Ce qui peut se présenter ainsi :

7 * 6 = ? (et nous comprenons qu'il faut remplacer le ? par le résultat).

Ayant appris nos tables de multiplication, nous connaissons « par coeur » la réponse :

7 * 6 = 42

*Autre exemple*

32 * 71 = ?

Comme on ne nous a pas appris la table de 71, ni celle de 32, le résultat ne vient pas « par coeur ». Sauf à tricher en utilisant un appareil à calculer, nous posons la multiplication, effectuons des multiplications partielles qui utilisent les tables, additionnons les retenues éventuelles et aboutissons à :

32 * 71 = 2272

*Mais il existe des situations où on associe plus de deux nombres*

Par exemple dans le programme nous avons quatre dates (converties en jours juliens) et pour chacune deux coordonnées équatoriales.

« Il s'est trouvé des mathématiciens » pour trouver astucieux de présenter ces informations sous la forme d'un tableau de quatre lignes comportant chacune trois valeurs. Par exemple :

| | | |
|---|---|---|
| jjulien1 | ad1 | dec1 |
| jjulien2 | ad2 | dec2 |
| jjulien3 | ad3 | dec3 |
| jjulien4 | ad4 | dec4 |

Ce tableau, que l'on encadre de deux crochets, est appelé une matrice.

Voir à ce propos :

http://maths.cnam.fr/IMG/pdf/Alg.1_calcul_matriciel.pdf

Dans le cas de Python avec l'extension Numpy, une matrice de deux lignes de trois nombres sera crée par exemple comme ceci :

```
a = np.array([[1, 2, 3],
              [4, 5, 6]])
```

Ayant créé ou hérité d'un concept nouveau la communauté des mathématiciens a démontré, imaginé des manières d'additionner, soustraire, multiplier, diviser, faire tourner… les matrices.
Cet outil étant adapté à la résolution de certains problèmes… on retrouve des fonctions de calcul sur/avec des matrices dans les langages informatiques modernes tels que Python (et son extension Numpy).
La fonction numpy appelée **.dot** réalise le produit de deux matrices.
(Voir dans la page https://www.courspython.com/tableaux-numpy.html le paragraphe consacré à numpy.dot()

### 3.8.1. Passer des coordonnées géocentriques ↔ écliptiques : lignes 65 à 71

Fonction appelée en ligne 328.

| *Code* | *Explication* |
|---|---|
| `def eclRot(vec):` | Paramètre : vec contient une matrice sur laquelle on veut effectuer une modification. |
| `EcRot = ...` | Matrice qui va multiplier vec |
| `return vec` | Retourne la matrice modifiée |

*Code de la fonction :*

```
#ecliptic rotation matrix
def eclRot(vec):
    ecRot = np.array([[1, 0, 0],
                      [0, cos(ecA), sin(ecA)],
                      [0, -sin(ecA), cos(ecA)]], dtype='float')
    vec = ecRot.dot(vec)
    return vec
```

## 3.9. Fonction affichant les résultats et proposant leur sauvegarde

### 3.9.1. Affichage des résultats : lignes 306 à 339

Cette partie de code découle de l'utilisation de la fonction print() avec Python3 et en montre la souplesse. Rien à dire : code linéaire et simple.

### 3.9.2. Sauvegarde éventuelle des résultats dans un fichier : lignes 340 à 348

Code simple, sans difficulté particulière.

# 4. LE CODE :
# CALCULS LIÉS AU FORMULAIRE DE GAUSS

Les lignes de code présentées jusqu'ici fournissent des « servitudes » au programme, mais ne sont pas directement liées au raisonnement imaginé par Gauss.

Dans ce qui suit, nous devons présenter, de la façon abordable par un nombre le moins réduit possible de lecteurs comment on peut « faire vivre par le calcul » un raisonnement mathématique.

Tâche quasi insurmontable…

Mais tant qu'on n'a pas tenté de faire, impossible de prétendre que c'était impossible;)

## 4.1. Transformation des données chargées depuis le fichier.

Celles -ci ont été commentées page 7.

### 4.1.1. Transformation en nombres utilisables : lignes 113 à 121

Les données initiales sont contenues dans le tableau **data** .

Les données transformées sont progressivement écrites dans le tableau **ndata** qui a été préalablement initialisé avec des zéros.

*À la fin,*

...tous les nombres sont des flottants (nombres à virgules) et les angles sont en radians (parce que Python ne pratique les fonctions trigonométriques qu'en radians et non en degrés et décimales).

*Exemple*

On a placé (momentanément) ceci en ligne 122

```
print(ndata)
```

et obtenu ce qui suit :

```
 [[  2.45756184e+06   0.00000000e+00   4.56831714e+00
5.37034417e-01
    -1.95340792e-02   9.32338427e-01   4.04140840e-01]
  [  2.45756981e+06   0.00000000e+00   4.48641102e+00
4.76777943e-01
    -1.53798333e-01   9.22112511e-01   3.99703505e-01]
  [  2.45757672e+06   0.00000000e+00   4.41517177e+00
4.02790382e-01
    -2.68247059e-01   8.99827935e-01   3.90044909e-01]
  [  2.45758265e+06   0.00000000e+00   4.35733374e+00
3.23198132e-01
```

```
-3.63521428e-01   8.71021804e-01   3.77563243e-01]]
```