Tutoriel : Apprendre l'Astronomie avec Python

Ahmed Ammar (`ahmed.ammar@fst.utm.tn`)

Faculté des Sciences, Université de Tunis El Manar et Société Astronomique de Tunisie.

2019

Table des matières

1 Python

Python est un langage de programmation moderne de haut niveau, orienté objet et d'usage général.

Caractéristiques générales de Python :

- Langage simple :
  - facile à lire et à apprendre avec une syntaxe minimaliste.
- Langage concis et expressif :
  - moins de lignes de code
  - moins de bugs
  - plus facile à maintenir.

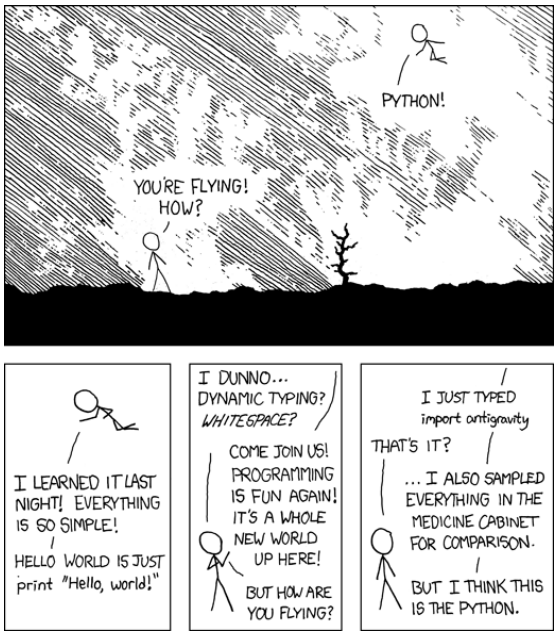
Détails techniques :

- Typé dynamiquement :
  - Pas besoin de définir le type des variables, les arguments ou le type des fonctions.
- La gestion automatique de la mémoire :
  - Aucune nécessité d'allouer explicitement et désallouer la mémoire pour les variables et les tableaux de données. Aucun bug de fuite de mémoire.
- Interprété :
  - Pas besoin de compiler le code. L'interpréteur Python lit et exécute le code python directement.

Avantages :

- Le principal avantage est la facilité de programmation, qui minimise le temps nécessaire pour développer, déboguer et maintenir le code.

— Langage bien conçu qui encourage les bonnes pratiques de programmation :
  — Modulaire et orientée objet, permet l'encapsulation et la réutilisation de code. Il en résulte souvent un code plus transparent, plus facile à améliorer et sans bug.
  — Documentation intégré avec le code.
— De nombreuses bibliothèques standards, et de nombreux packages add-on.



2 Installation d'un environnement Python scientifique

2.1 Installation sur ordinateur

Qu'est ce que Anaconda ? L'installation d'un environnement Python complet peut-être une vraie galère. Déjà, il faut télécharger Python et l'installer. Par la suite, télécharger un à un les packages dont on a besoin. Parfois, le nombre de ces librairies peut-être grand.

Par ailleurs, il faut s'assurer de la compatibilité entre les versions des différentes packages qu'on a à télécharger. Bref, ce n'est pas amusant.

Anaconda est une distribution Python. A son installation, Anaconda installera Python ainsi qu'une multitude de packages (voir liste de packages anaconda).

Cela nous évite de nous ruer dans les problèmes d'incompatibilités entre les différents packages.

Finalement, Anaconda propose un outil de gestion de packages appelé conda. Ce dernier permettra de mettre à jour et installer facilement les librairies dont on aura besoin pour nos développements.

Préparer la formation : téléchargement d'Anaconda. Nous demandons à tous les étudiants de télécharger Anaconda. Pour cela, il faut télécharger un installeur à partir de `https://www.anaconda.com/download/`, correspondant à votre système d'exploitation (Windows, Mac OS X, Linux). Il faut choisir entre 32 bits ou 64 bits (pour la version *Python 3*) selon que votre système d'exploitation est 32 bits ou 64 bits.

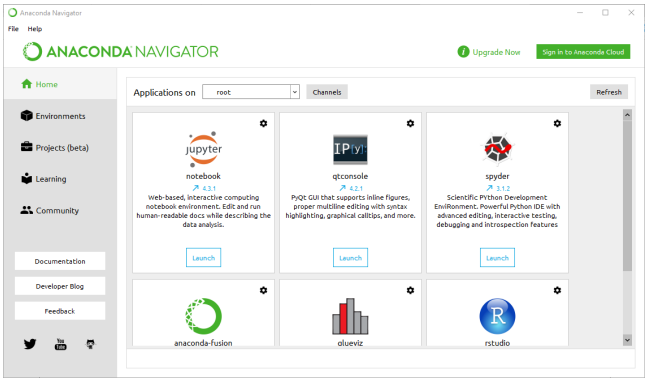


FIGURE 1 – Interface graphique du navigateur Anaconda sur Windows

Remarque
Anaconda installe plusieurs exécutables pour développer en Python dans le répertoire *anaconda/bin*, sans toujours créer des raccourcis sur le bureau ou dans un menu. Nous nous occuperons au tout début de la formation de créer des raccourcis pour pouvoir lancer l'application web *Jupyter notebook*. Vous pouvez lancer le notebook depuis le navigateur Anaconda.

2.2 Installation sur smartphone

Pydroid 3 - IDE éducatif pour Python 3. Pydroid 3 est l'IDE éducatif Python 3 le plus simple et le plus puissant à utiliser pour Android.

Pydroid 3 fournit :

— Interpréteur Python 3.6 hors connexion : Internet n'est pas nécessaire pour exécuter des programmes Python.

— Pip package manager et un référentiel personnalisé pour les packages de roues prédéfinis pour les bibliothèques scientifiques améliorées, tels que numpy, scipy, matplotlib, scikit-learn et jupyter.
— ...

Installer et utiliser Pydroid 3 sur son smartphone : Pydroid est une application Android que vous pouvez obtenir sur Google Play : https://play.google.com/store/apps/details?id=ru.iiec.pydroid3

Les étapes suivantes, dans les figures ci-dessous, vous permettent d'utiliser le cahier Jupyter sur votre téléphone portable n'importe où et à tout moment pour vous entraîner au maximum et vous familiariser avec tous les exemples de programmation de ce cours.

Phase installation :

1. Installer Pydroid 3 depuis Google Play : https://play.google.com/store/apps/details?id=ru.iiec.pydroid3

2. Ouvrez l'application, sur le menu cliquez sur pip et allez à l'onglet "QUICK INSTALL" pour obtenir les bibliothèques scientifiques nécessaires à ce cours.

3. Dans "QUICK INSTALL", installer les packages *Jupyter* , *numpy* et *matplotlib*.

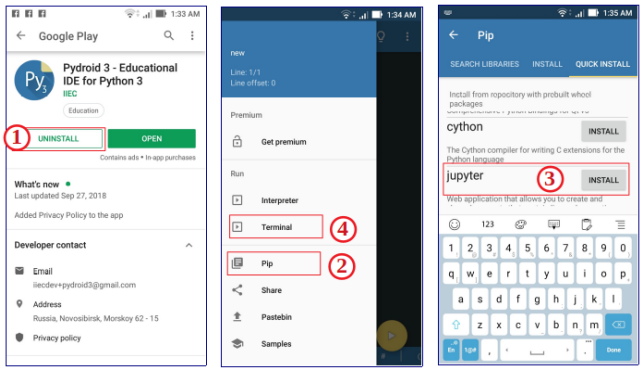


FIGURE 2 – Pydroid 3 : Phase installation

Phase utilisation :

4. Retournez au menu et ouvrez le **terminal**.

5. Sur le terminal, entrez la commande suivante :

```
jupyter notebook
```

6. Jupyter s'exécutera sur votre navigateur Web. Accédez au répertoire dans lequel vous avez des notebooks à ouvrir, à télécharger (bouton *upload*) ou à créer (bouton *New*).

7. Amusez-vous à travailler sur le notebook : créez du contenu, lancez et modifiez des exemples

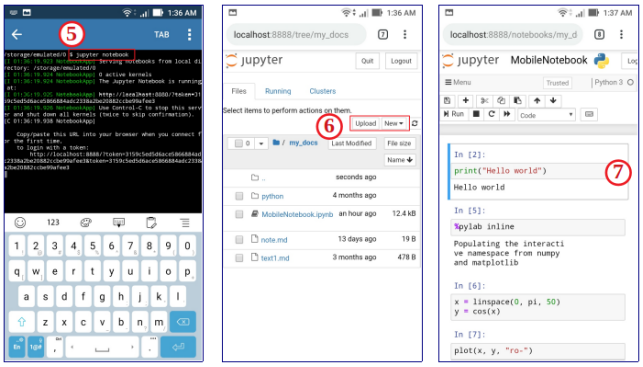


FIGURE 3 – Pydroid 3 : Phase utilisation

3 PyEphem

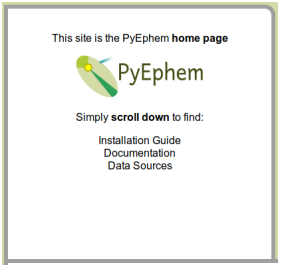
Le module **PyEphem** (`https://rhodesmill.org/pyephem/`) vous permet d'effectuer de nombreux calculs astronomiques professionnels. Vous pouvez l'utiliser :

- calculer les positions du soleil, de la lune, des planètes et de leurs lunes.
- déterminer l'emplacement actuel des astéroïdes, des comètes et des satellites artificiels (fournir des éléments orbitaux).
- déterminer le lever et le coucher du soleil d'objets divers en fonction de la position de l'observateur à la surface de la Terre.
- trouver dans quelle constellation l'objet est.
- calculer la position de étoiles, galaxies, etc.

PyEphem est basé sur des procédures écrites pour le programme **XEphem** (`www.clearskyinstitute.com/xephem/`), disponible depuis de nombreuses années pour les utilisateurs du système *Unix*.

3.1 Installation

La version 3.7.6.0 est la version la plus récente de PyEphem.



Le moyen le plus simple d'installer PyEphem sur un ordinateur Window, Linux ou Mac OS, après s'être assuré que la distribution Python **Anaconda** est bien installée, est d'utiliser la commande `pip`, comme ceci :

```
$ pip install pyephem
```

3.2 Importation de module et lieu de l'observateur

Le module **PyEphem** est disponible sous le nom **ephem** qui peut être modifié lors de l'importation par l'alias **ep** pour raccourcir la programmation.

```
import ephem as ep
#OBSERVATEUR
obs = ep.Observer()
# nous créons une structure dans laquelle des données
# sur la position de l'observateur seront stockées
obs.lon = "10.00"
obs.lat = "36.5"
obs.name = "SAT-Tunis"
obs.elevation = 100.0
```

Les fonctions et les structures du module `ephem` sont appelées dans la fonction `ep.method()`. Tout d'abord, nous définissons la position de l'observateur. Pour ce faire, nous créons la structure appropriée (`structure = ep.Observer()`) et remplissons ses champs (`structure.champ = valeur`).

3.3 Objets célestes et leur localisation

Tous les objets célestes importants tels que le Soleil, la Lune, les planètes et leurs lunes peuvent être créés par la structure de `fonction = ep.NomObjet()`.

```
# Objets
# nous créons une structure dans laquelle les données seront stockées
lune = ep.Moon()
```

Après avoir créé l'objet, nous pouvons calculer son emplacement actuel en fournissant des informations sur l'observateur, situées dans la structure `obs` précédemment créée.

```
# CALCULS
# on calcule la position actuelle de l'objet
# pour l'observateur créé plus tôt
lune.compute(obs)
```

Comme vous pouvez le constater, vous pouvez définir plusieurs observateurs différents et compter les coordonnées de l'objet sélectionné pour différents endroits de la Terre. Cela peut être utile lors de la planification de campagnes d'observation menées par différents observatoires.

3.4 Lecture de coordonnées

Les coordonnées calculées sont lues à partir des champs de structure d'objet.

```
# coordonnées calculées
print("Position actuelle de la Lune")
print(" ------------------------------ ")
# nous affichons l'ascension droite et la déclinaison
print("RA : ", lune.ra)
print("Dec : ", lune.dec)
# nous affichons l'azimut et l'élévation
print("--------------------------------")
print("Az : ", lune.az)
print ("El : " , lune.alt)
```

Les valeurs calculées sont données au format ***heures : minutes : secondes*** ou ***degrés : minutes : secondes d'arc*** pour l'heure actuelle UT sur époque 2000.

```
Position actuelle de la Lune
 ------------------------------
RA :  6:18:27.81
Dec :  20:49:04.9
--------------------------------
Az :  197:06:32.3
El :  56:50:39.8
```

3.5 Coordonnées en degrés

Toutes les coordonnées calculées par les procédures du module **PyEphem** sont données en **radians**. lorsque nous voulons écrire leur valeur, par exemple, les radians sont automatiquement convertis au format approprié (heures, minutes, secondes dans le cas d'une ascension droite ou degrés, minutes, secondes d'arc pour d'autres coordonnées).

Si vous souhaitez utiliser des coordonnées calculées sur un graphique, il est utile de les convertir en degrés à l'aide de la fonction degrés.

```
# coordonnées azimutales en degrés sous forme d'un nombre réel
print(" -------------------------------- ")
print("Az (deg): ", degrees(lune.az))
print("El (deg): ", degrees(lune.alt))
```

Azimut et élévation en degrés :

```
--------------------------------
Az (deg):  207.85208210454263
El (deg):  55.334644372169485
```

3.6 Date et heure

Nous pouvons attribuer n'importe quelle date et heure à chaque observateur :

```
# PROPRE DATE ET HEURE TU
obs.date = "2019/01/13 10:00:00"
```

Il faut seulement se rappeler que les chiffres de la date sont séparés par le signe / et l'heure par deux points.

3.7 Mouvement apparent du Soleil - le début du programme

```
# IMPORTATION
from pylab import *
import ephem as ep
# OBSERVATEUR
obs = ep.Observer()
# TUNIS
obs.lon, obs.lat, obs.elev = '10.08', '36.4', 100.0
obs.name= "SAT-TUNIS"
# OBJET
soleil = ep.Sun()
# TEMPS
tm = linspace(0 , 24 , 25)
# GRAPHIQUE
pt = subplot(111 , polar= True )
```

3.8 Mouvement apparent du Soleil - solstice d'été

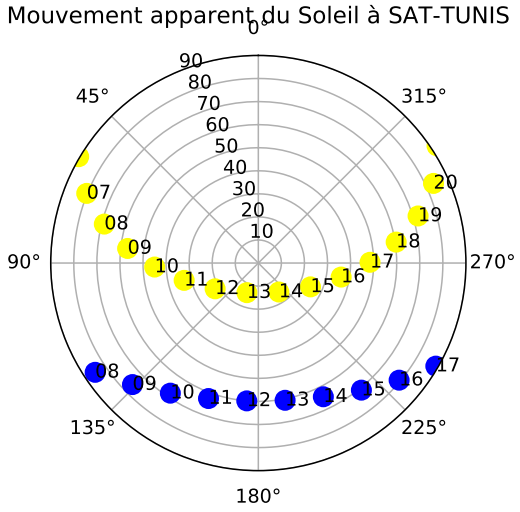
```
# BOUCLE PRINCIPALE
for t in tm :
    # changement de temps
    obs.date = "2014/06/21 %02 d :00:00 "%t
    # on calcule les coordonnées
    sun.compute(obs)
    # coordonnées azimutales - azimut en radians
    az = float(repr(sun.az))
    el = degrees(float(repr(sun.alt)))
    # graphique - on change l'élévation par une distance zénithale
    pt.plot([az], [90 - el], ls =" ", marker= " o ", c =" yellow ", \
    markersize =10)
    # heure locale UTC +2 heures en été
    if el > 0:
        pt.text (az, 90 - el, " %02 d "%(t+2), fontsize =10, \
        ha = 'left' , va = 'center')
```

3.9 Mouvement apparent du Soleil - solstice d'hiver

```
    # TRANSFERT HIVERNAL - nous répétons les calculs "en décembre"
    obs.date = "2014/12/22 %02d:00:00" % t
    soleil.compute(obs)
    az = float(repr(soleil.az))
```

```
    el = degrees(float(repr(soleil.alt)))
    pt.plot([az], [90 - el], ls ="", marker= "o", c ="blue", \
    markersize =10)
    # heure locale UTC +1 heures en hiver
    if el > 0:
        pt.text (az, 90 - el, "%02d"%(t+1), fontsize =10, \
        ha = 'left' , va = 'center')
# nous limitons la distance zénithale à 90 degrés - horizon
plt.set_rmax(90.0)
# nous plaçons le nord en haut du graphique
plt.set_theta_zero_location("N")
# ENGISTREZ LE GRAPHIQUE EN FORMAT PDF ET PNG
plt.savefig("figs/mvtSoleil.pdf"); plt.savefig ("figs/mvtSoleil.png")
plt.show ()
```

3.10 Mouvement apparent du Soleil - le graphique



Exercise 1 : Mouvement apparent du Soleil - région polaire

Changer les coordonnées géographiques. Cette fois dans une ville proche de la région polaire et tracez le nouveau graphique. Que s'est-il passé et comment décrivez-vous le phénomène ?

3.11 Mouvement rétrograde de Mars

Dès l'Antiquité, il a été remarqué que le mouvement de certains objets dans le ciel est d'un caractère différent du trafic de la majorité (**étoiles fixes**). De tels objets ont été appelés étoiles errantes (**planètes**). Aujourd'hui, nous savons que

errer sur les planètes a pour effet de placer leur mouvement en même temps que le mouvement de la Terre autour du Soleil. Nous pouvons retracer ce phénomène sur l'exemple de Mars.

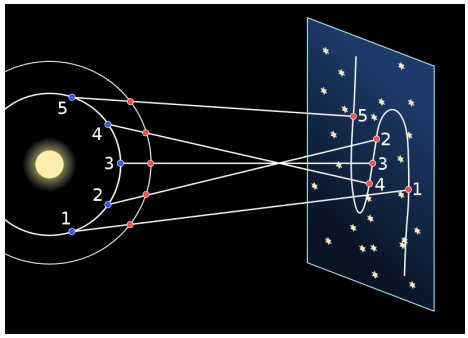


FIGURE 4 – Projetion depuis la Terre (en bleu) des mouvements de la planète extérieure (en rouge) sur la sphère des étoiles fixes. *Source : Wikipédia*

3.12 Mouvement rétrograde de Mars - début du programme

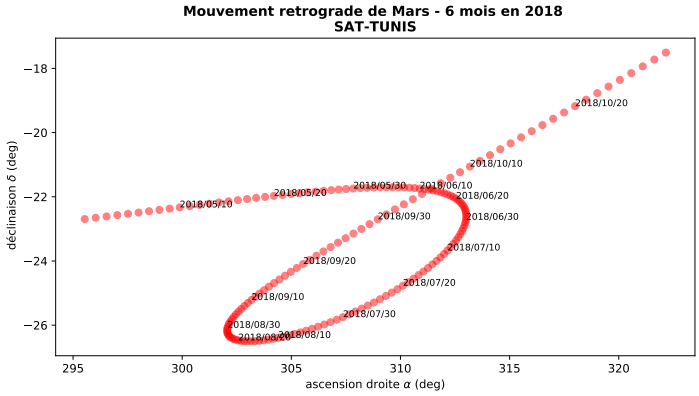
```
# IMPORTATION
import ephem as ep
# deux fonctions supplémentaires du module datetime sont nécessaires
from datetime import datetime , timedelta

# OBSERVATEUR
obs = ep.Observer()
# COORDONNÉES DE TUNIS
obs.lon, obs.lat, obs.elev = '10.08', '36.4', 100.0
obs.name = "SAT-TUNIS"
# MARS
mr = ep.Mars()
```

3.13 Mouvement rétrograde de Mars - boucle principale

```
from pylab import *
plt.figure(figsize=(10, 5))
for i in range (0 , 181):
    # nous changeons la date d'un jour pendant six mois
    dt = datetime (2018, 5, 1) + timedelta(i)
    ds = "%d/%02d/%02d"%(dt.year, dt.month, dt.day)
    print(" jour de l'année: ", i +1 , ds)
    # fixer la date de l'observateur et calculer les coordonnées
    obs.date = ds
    mr.compute(obs)
    ra = degrees(float(repr(mr.ra)))
    de = degrees(float(repr(mr.dec)))
    # on dessine des objets
    plot([ra], [de], c = "red", marker = "o", alpha =.5)
```

```
    # nous ajoutons une description de la date en moyenne tous les 10 jours
    if (dt.day % 10) == 0: text(ra, de, ds, fontsize =8)

# description du graphique
xlabel("ascension droite " + r"$\alpha$ (deg)")
ylabel(" déclinaison " + r"$\delta$ (deg)")
title("Mouvement retrograde de Mars - 6 mois en 2018 \n"+obs.name, fontweight='bold')
savefig("../figs/retrogradeMars.pdf"); savefig("../figs/retrogradeMars.png")
show()
```

3.14 Mouvement rétrograde de Mars - le graphique

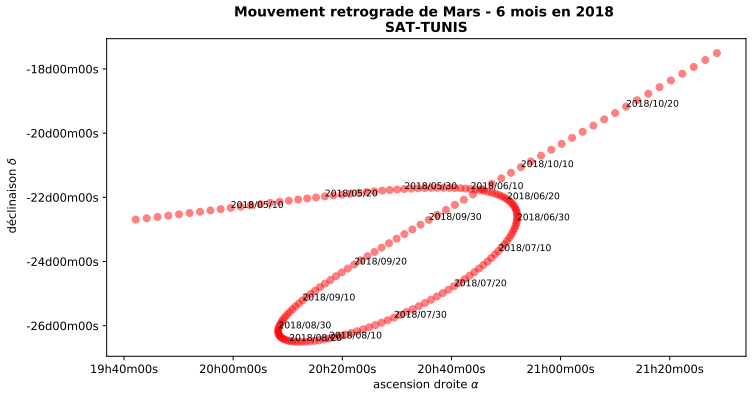


3.15 Mouvement rétrograde de Mars - changement du texte des axes

```
# conversion RA donné en degrés
# sur les formats heure, minute et seconde
def RAd2hms (x, loc):
    h = x//15
    m = int(((x - h * 15.0) / 15.0) * 60.0)
    s = ((x - h *15 - m / 4.0) / 15.0) * 3600.0
    return "%02dh%02dm%02ds"%(h, m, s)
# changement de déclinaison donné en degrés
# le format du degré, minute, second arc
def DEd2dms (x , loc ):
    d = int(fabs(x))
    m = int((fabs(x) - d)*60)
    s = (fabs(x) - d - m /60.0)*3600.0
    if x <0: d = -1 * d
    return " %02dd%02dm%02ds"%(d, m, s)

# description du graphique
xlabel("ascension droite " + r"$\alpha$")
gca().xaxis.set_major_formatter(FuncFormatter(RAd2hms))
ylabel(" déclinaison " + r"$\delta$")
gca().yaxis.set_major_formatter(FuncFormatter(DEd2dms))
```

```
title("Mouvement retrograde de Mars - 6 mois en 2018 \n"+obs.name, fontweight='bold')

savefig("../figs/retrogradeMars.pdf"); savefig("../figs/retrogradeMars.png")
show()
```

3.16 Mouvement rétrograde de Mars - le graphique



Exercise 2 : Quand le prochain mouvement rétrograde de la planète Mars ?

Changez la période et trouvez le prochain mouvement rétrograde de la planète Mars.

3.17 Mouvement des lunes galiléennes - début du programme

```
# IMPORTATION
import ephem as ep
# NOUS CRÉONS LES OBJETS
Io = ep.Io()
Eu = ep.Europa()
Ga = ep.Ganymede()
Ca = ep.Callisto()
# Créons des tableaux vide pour
# SAUVEGARDER LES COORDONNÉES
y = []
xIo = []
xEu = []
xGa = []
xCa = []
```

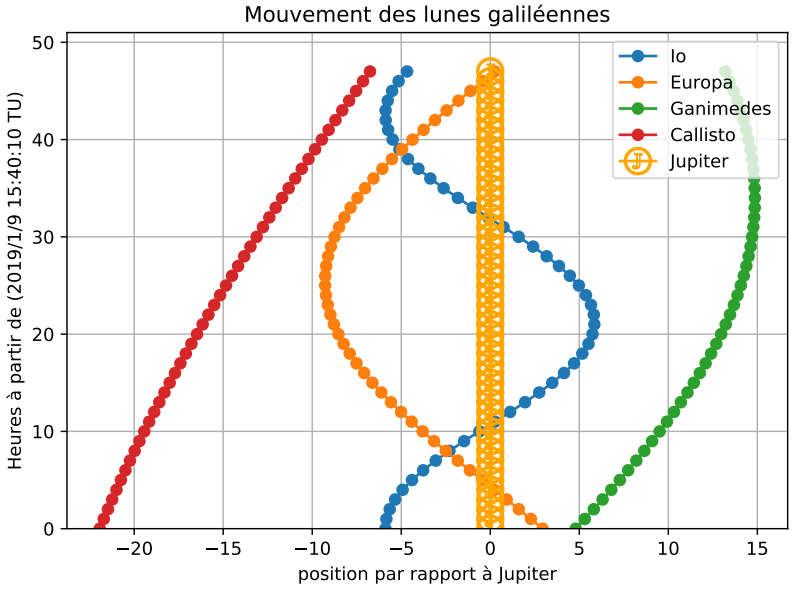
3.18 Mouvement des lunes galiléennes - boucle principale

```
# pas de temps - heure
dt = ep.hour
# temps initial
ts = ep.now()
# heure actuelle
tm = ts
N=2*24
for i in range(N):
    # nous calculons des valeurs y
    y.append((tm - ts)*24.0)
    # nous calculons des valeurs x
    Io.compute(tm)
    Eu.compute(tm)
    Ga.compute(tm)
    Ca.compute(tm)
    # nous ajoutons des calculs aux tableaux
    xIo.append(Io.x)
    xEu.append(Eu.x)
    xGa.append(Ga.x)
    xCa.append(Ca.x)
    # on augmente le temps d'une heure
    tm += dt
```

3.19 Mouvement des lunes galiléennes - le graphique

```
from pylab import *
fig1 = plt.figure()
plot(xIo, y, marker ="o", label ="Io")
plot(xEu, y, marker ="o", label ="Europa")
plot(xGa, y, marker ="o", label ="Ganimedes")
plot(xCa, y, marker ="o", label ="Callisto")
plot(zeros(len(y)),y ,marker =r"$\mathcircled{J}$", markersize =15,
     label ="Jupiter",color ="orange", fillstyle= 'none')
xlabel(u"position par rapport à Jupiter")
ylabel("Heures à partir de (%s TU)"% ts)
ylim(0,N+3)
title(u"Mouvement des lunes galiléennes")
grid()
legend(loc =1)

tight_layout()
savefig("../figs/mvtjov.pdf"); savefig("../figs/mvtjov.png")
show()
```

3.20 Mouvement des lunes galiléennes - le graphique



3.21 Détermination de l'éclipse solaire - début du programme

```
# IMPORTATION
import ephem as ep
# OBSERVATEUR
obs = ep.Observer()
obs.lon, obs.lat, obs.elev = '10.08', '36.4', 100.0
obs.name = "SAT-TUNIS"
# OBJETSNIS
soleil = ep.Sun()
lune = ep.Moon()
# pas de temps - heure
dt = ep.hour
# temps initial
#ts = ep.now()
ts=ep.Date("2015-01-01 00:00:00")
# heure actuelle
tm = ts
```

3.22 Détermination de l'éclipse solaire - boucle principale

Nous vérifions la séparation du soleil et de la lune toutes les heures pendant les 10 prochaines années.

```
for i in range (365*24*10):
    # nous fixons l'heure actuelle
    obs.date = tm
    # nous calculons les coordonnées
    soleil.compute(obs)
    lune.compute(obs)
    # rayons
    rs = soleil.radius
    rl = lune.radius
    # on calcule la distance angulaire
    sp = ep.separation(soleil, lune)
    # on vérifie si la somme des rayons sera inférieure
    # à la séparation calculée
    if sp < rs + rl :
    # nous vérifions si le soleil sera au-dessus de l'horizon
        if soleil.alt > 0.0:
            print("Date de l'eclipse UT: ", ep.Date(tm), "Separation: ", sp)
    # on augmente le temps par pas d'une heure
    tm += dt
```

3.23 Détermination de l'éclipse solaire -

Les dates de l'éclipse solaire planifiée visible de la Tunisie au cours des 10 prochaines années.

```
Date de l'eclipse UT:  2020/6/21 05:00:00 Separation:  0:30:50.0
Date de l'eclipse UT:  2022/10/25 10:00:00 Separation:  0:29:25.7
Date de l'eclipse UT:  2022/10/25 11:00:00 Separation:  0:30:51.9
Date de l'eclipse UT:  2025/3/29 11:00:00 Separation:  0:32:49.2
Date de l'eclipse UT:  2026/8/12 18:00:00 Separation:  0:20:53.8
Date de l'eclipse UT:  2027/8/2 08:00:00 Separation:  0:30:44.7
Date de l'eclipse UT:  2027/8/2 09:00:00 Separation:  0:04:33.3
Date de l'eclipse UT:  2027/8/2 10:00:00 Separation:  0:21:01.3
Date de l'eclipse UT:  2028/1/26 16:00:00 Separation:  0:26:09.8
```

3.24 Phases d'une éclipse solaire - début du programme

```
# IMPORTATION
from pylab import *
import ephem as ep
# OBSERVATEUR
obs = ep.Observer()
# TUNIS
obs.lon, obs.lat, obs.elev = '10.08', '36.4', 100.0
obs.name= "SAT-TUNIS"
# CRÉER DES OBJETS
soleil = ep.Sun()
lune = ep.Moon()
# intervalle de temps - 20 minutes
dt = ep.hour/4.
# TEMPS DE DÉBUT
ts = ep.Date("2027-08-02 08:00:00")
obs.date = ts
# nous calculons les coordonnées
```

```
soleil.compute(obs)
lune.compute(obs)
rs = degrees(soleil.radius)
rl = degrees(lune.radius )
```

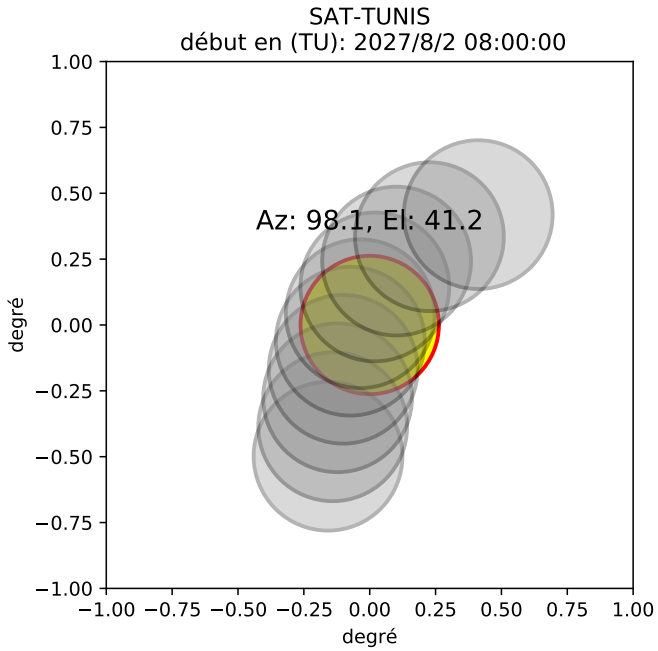
3.25 Phases d'une éclipse solaire - suite

```
# nous calculons les coordonnées
soleil.compute(obs)
lune.compute(obs)
rs = degrees(soleil.radius)
rl = degrees(lune.radius )

# Nous créons un graphique en l'attribuant à un pl COMME variable
fig=plt.figure(figsize=(6,5))
pl = subplot(111, aspect ="equal")
# titre du graphique avec date et heure
pl.set_title (obs.name+"\n début en (TU): "+str(ep.Date(ts)))
# Nous nous plaçons le soleil dans le centre
sc=Circle((0 ,0), rs, facecolor ="yellow",
            edgecolor ="red", lw =2)

pl.add_artist(sc)
# les coordonnées du Soleil
pl.text(0, rs+0.1, "Az: %.1f, El: %.1f"%(degrees(soleil.az), degrees(soleil.alt)),
        ha='center', fontsize =14)
```

3.26 Phases d'une éclipse solaire - boucle principale

```
# Nous plaçons la Lune dans la figure
for i in range(10):
    print("time UT: ", ep.Date(ts))
    obs.date = ts
    # nous calculons les coordonnées
    soleil.compute(obs)
    lune.compute(obs)
    # Nous calculons la différence de position
    az = degrees(soleil.az - lune.az)
    el = degrees(soleil.alt - lune.alt)
    # dessiner et la position réelle en empillements d'image de Lune; le Soleil estau centre
    kc = Circle((az , el), rl , facecolor ="gray",
                edgecolor ="black", lw =2, alpha =0.3)
    pl.add_artist(kc)
    # augmenter le temps de 20 minutes
    ts += dt

pl.set_xlim (-1.0, 1.0)
pl.set_ylim (-1.0, 1.0)
pl.set_xlabel ("degré")
pl. set_ylabel ("degré")
plt.tight_layout()
savefig("../figs/eclipse_sol_" + str(ts) +".pdf"); savefig("../figs/eclipse_sol_" + str(ts) +".png")
show ()
```

3.27 Phases d'une éclipse solaire - le graphique



3.28 Conjonction de la lune et des planètes

```
# IMPORTOWANIE
from pylab import *
import ephem as ep
# OBSERVATEUR
obs = ep.Observer()
# TUNIS
obs.lon, obs.lat, obs.elev = '10.08', '36.4', 100.0
obs.name= "SAT-TUNIS"
# NOUS CRÉONS UN OBJET
# on vérifie si c'est sous l'horizon
soleil = ep.Sun()
lune = ep.Moon()
venus = ep.Venus()
mars = ep.Mars()
jupiter = ep.Jupiter()
# pas de temps - heure
dt = ep.hour
# temps initial
ts = ep.now()
# heure actuelle
tm = ts
```

3.29 Conjonction de la lune et des planètes - boucle principale

```
# BOUCLE PRINCIPAL DU PROGRAMME
for i in range (365*24*1):
# nous fixons l'heure actuelle
    obs.date = tm
    # nous calculons des coordonnées
    soleil.compute(obs)
    lune.compute(obs)
    venus.compute(obs)
    mars.compute(obs)
    jupiter.compute(obs)
    # on calcule la séparation
    s1 = ep.separation(venus , lune)
    s2 = ep.separation(mars , lune)
    s3 = ep.separation(jupiter , lune)
    # la séparation doit être inférieure à 5 degrés
    if degrees(s1) < 5:
        # nous vérifions si la lune sera au-dessus de l'horizon
        # et si le soleil est au-dessous de l'horizon
        if degrees(lune.alt) > 5.0 and degrees(soleil.alt) < -5.0:
            print("-----------------------------------------------------------------")
            print(u"précédente nouvelle lune , UT :", ep.previous_new_moon(ep.Date(tm)))
            print(u"Vénus - Lune , UT :", ep.Date(tm) ,"séparation :", s1)
            print(u"pos. Lune :", lune.az ,"El :", lune.alt)
```

3.30 Conjonction de la lune et des planètes - boucle principale (suite)

```
if degrees(s2) < 5:
    if degrees(lune.alt) > 5.0 and degrees(soleil.alt) < -5.0:
        print("-----------------------------------------------------------------")
        print(u"précédente nouvelle lune , UT :", ep.previous_new_moon(ep.Date(tm)))
        print(u"Mars - Lune , UT :", ep.Date(tm) ,"séparation :", s2)
        print(u"Pos. Lune, Az :", lune.az ,"El :", lune.alt)
if degrees(s3) < 5:
    if degrees(lune.alt) > 5.0 and degrees(soleil.alt) < -5.0:
        print("-----------------------------------------------------------------")
        print(u"précédente nouvelle lune , UT :", ep.previous_new_moon(ep.Date(tm)))
        print(u"Jupiter - Lune , UT :", ep.Date(tm) ,"séparation :", s3)
        print(u"Pos. Lune, Az :", lune.az ,"El :", lune.alt)
    # on augmente le temps d'une heure
tm += dt
```

Exemples de résultats de calculs :

```
-----------------------------------------------------------------
précédente nouvelle lune , UT : 2019/1/6 01:28:11
Jupiter - Lune , UT : 2019/1/31 03:47:21 séparation : 3:09:07.3
Pos. Lune, Az : 122:58:36.9 El : 8:18:03.0
-----------------------------------------------------------------
précédente nouvelle lune , UT : 2019/2/4 21:03:35
Vénus - Lune , UT : 2019/3/3 04:47:21 séparation : 4:07:18.5
pos. Lune : 121:01:33.3 El : 6:15:18.7
-----------------------------------------------------------------
```

4 Cartopy

Cartopy est un package Python conçu pour le traitement des données géospatiales afin de produire des cartes et d'autres analyses de données géospatiales.

Installation de Cartopy :

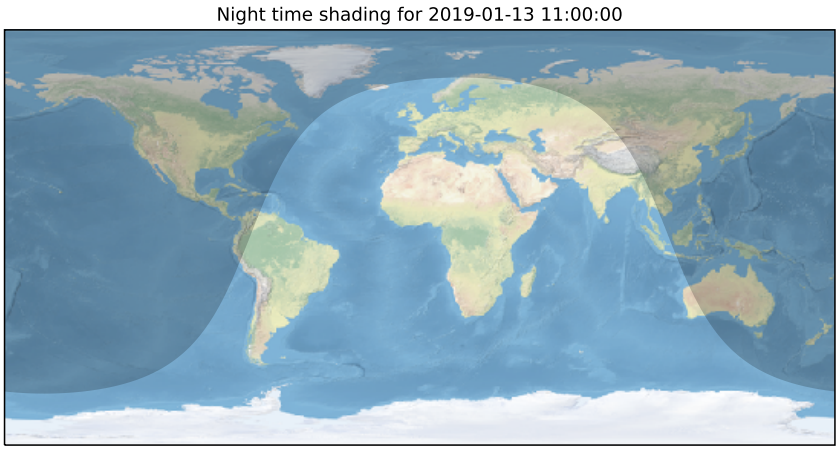
Le moyen le plus simple d'installer **Cartopy** est **Conda**. Pour toutes les plateformes, l'installation de cartopy peut se faire avec :

```
conda install -c conda-forge cartopy
```

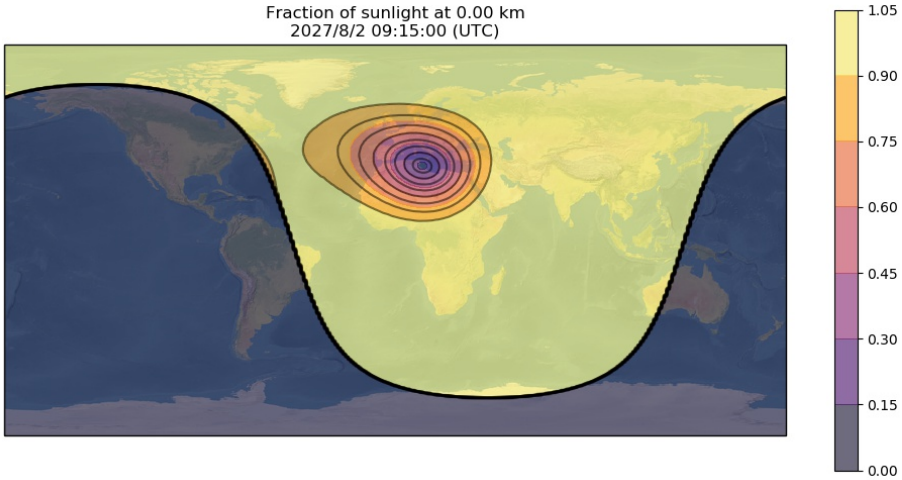
Exemple simple :

```
import datetime
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
from cartopy.feature.nightshade import Nightshade
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
date = datetime.datetime(2019, 1, 13, 11)
ax.set_title('Night time shading for {}'.format(date))
ax.stock_img()
ax.add_feature(Nightshade(date, alpha=0.2))
plt.show()
```

4.1 Graphique



4.2 Application : Eclipse totale du Soleil en 2027



Merci de votre attention !