TP de Physique n°19
« Houston, we've had a problem »

Objectifs de ce TP :

- Utiliser Python pour calculer la trajectoire d'un vaisseau spatial dans le champ gravitationnel de la Terre, puis de l'ensemble Terre-Lune. Dans le deuxième cas, vérifier qu'il existe des trajectoires, dites « trajectoires de retour libre », qui permettent d'utiliser la gravitation de la lune pour retourner sur Terre sans utiliser de carburant.

- Vérifier numériquement les résultats du cours sur les forces centrales Newtoniennes.

Introduction : La mission Apollo 13 : *(extrait de Wikipedia)*

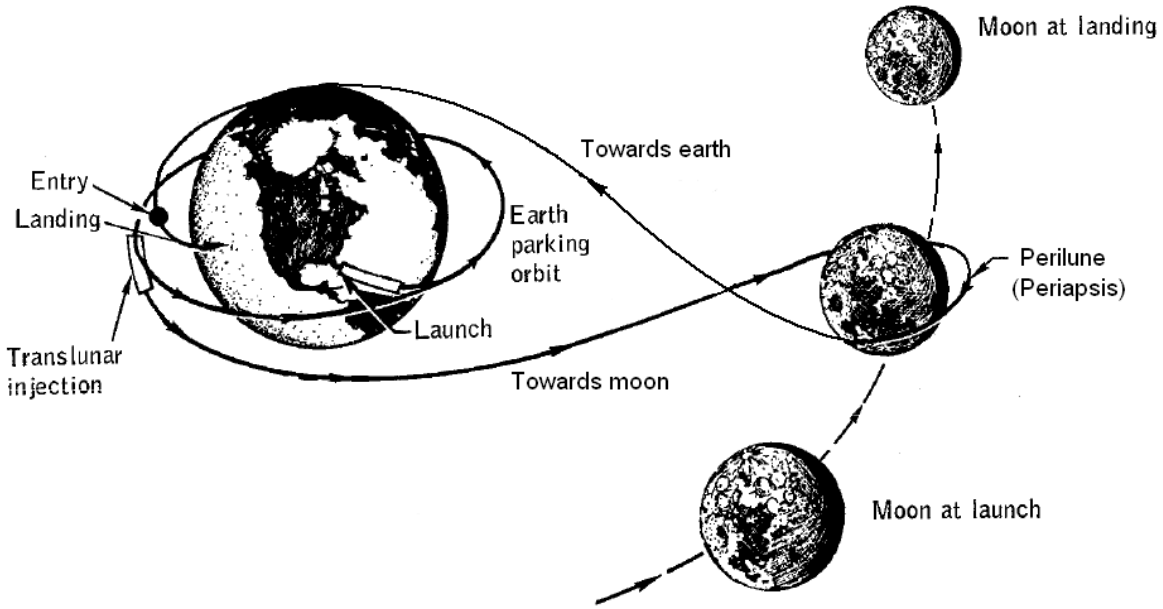
Apollo 13 (11 avril 1970 - 17 avril 1970) est une mission lunaire habitée du programme Apollo interrompue à la suite de l'explosion d'un réservoir d'oxygène au cours du trajet vers la Lune.
Ne pouvant faire demi-tour, le vaisseau et son équipage furent obligés de poursuivre leur voyage jusqu'à la Lune et d'en faire le tour pour utiliser son attraction gravitationnelle afin de revenir vers la Terre.
Le module de commande et de service Apollo étant devenu inhabitable, l'équipage se réfugia dans le module lunaire *Aquarius*, qui n'avait pas été prévu pour une occupation prolongée par l'ensemble de l'équipage. Malgré les très nombreux problèmes qu'il a fallu résoudre dans l'urgence, l'équipage put être récupéré sain et sauf !

Comme les missions antérieures, Apollo 13 devait permettre de ramasser des roches lunaires et de réaliser diverses expériences sur la Lune.
Le 14 avril 1970 à 03:07:53, un réservoir d'oxygène explose à 321 860 km de la Terre. Le directeur de vol Gene Kranz demande alors de fermer les valves des piles à combustibles pour arrêter la fuite, mais cette procédure ne sert à rien.
Les trois hommes qui constituent l'équipage passent alors dans le module lunaire « Aquarius », qui jouera le rôle de canot de sauvetage jusqu'à leur récupération. Aquarius n'a pas de siège et n'est pas prévu pour trois hommes mais ses dispositifs (notamment l'approvisionnement en électricité et en oxygène) lui garantissent provisoirement l'autonomie pour le retour sains et saufs des trois astronautes. De plus, seul le moteur d'Aquarius peut réaliser les corrections de trajectoire pour le retour, celui du module de commande « Odyssey » étant inutilisable.

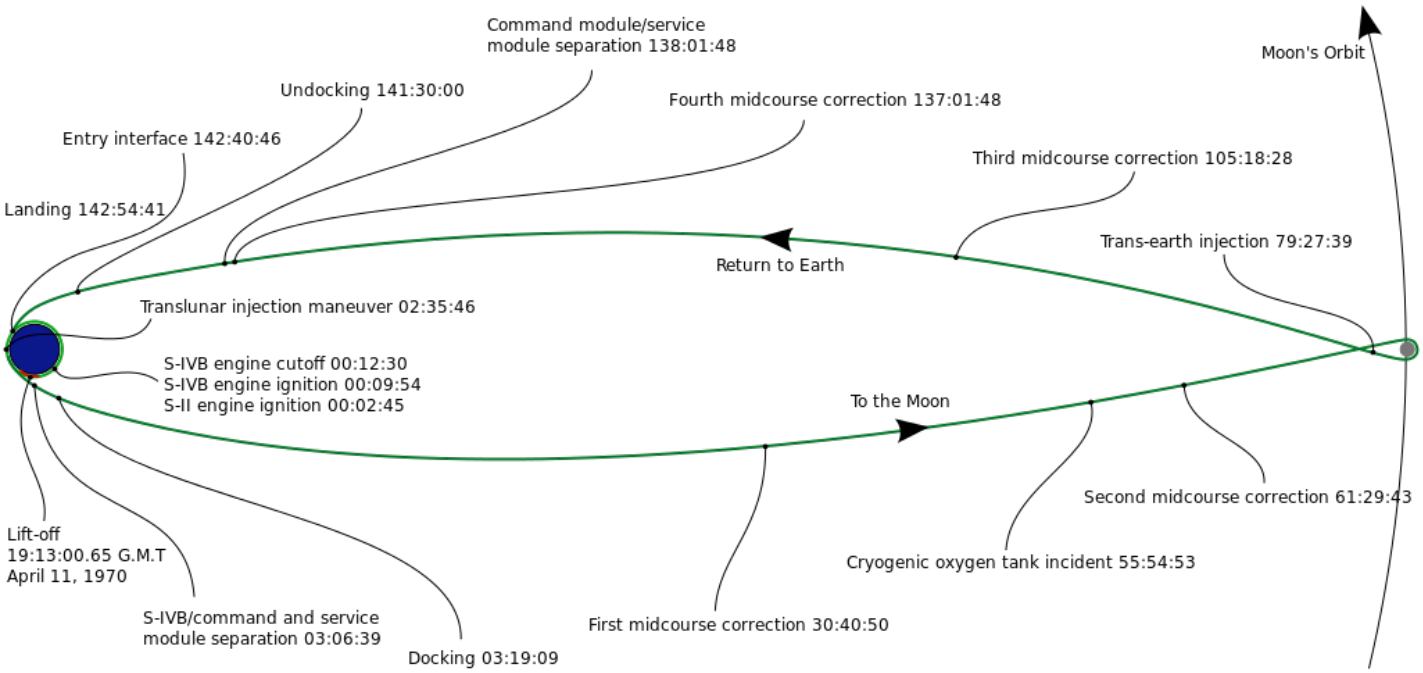
Afin d'économiser de l'énergie pour la rentrée dans l'atmosphère, 80 % des dispositifs électriques sont coupés, notamment les systèmes de guidage et l'« Environment Control System » qui assure le chauffage par des radiateurs, ce qui fait chuter la température du module à 9 °C. Seuls les appareils permettant la communication avec Houston sont maintenus allumés.
D'autre part, les trois hommes se trouvent confrontés à un autre problème: le taux de CO_2 (qui est toxique à fortes doses) augmente dans l'Aquarius, car ce dernier a été conçu pour deux hommes et non trois. Ils bricolent alors un adaptateur (avec du scotch !), qui leur permet d'installer des cartouches de rechange sur le filtre à air du module lunaire pour éliminer l'excédent de CO_2.

Pour la rentrée dans l'atmosphère, ils se réinstallent dans « Odyssey », seul équipé d'un bouclier thermique, et réactivent étape par étape les systèmes du module. Après l'amerrissage dans les eaux du Pacifique, à six kilomètres du porte-avion USS *Iwo Jima*, l'hélicoptère Sikorsky SH-3 Sea King récupère les trois hommes et les transporte au porte-avion qui les ramène à terre.

L'expression « *Houston, we've had a problem* » qui fut prononcée par l'astronaute Jack Swigert avec un calme impressionnant (en sachant qu'une panne à 300 000 mètres de la Terre a de fortes chances d'être fatale), est vite entrée dans la culture américaine.



Principe d'une « trajectoire de retour libre » qui permet à un vaisseau spatial d'utiliser la force de gravitation de la Lune pour être propulsé à nouveau sur la Terre.



Trajectoire de la mission Apollo 13, détaillant les divers évènements qui se sont produits.

I Rappel d'informatique : utilisation de la fonction `odeint` du module `scipy.integrate` pour la résolution d'équations différentielles avec Python :

1) Cas d'une équation du premier ordre :

On a déjà vu en TP info comment obtenir numériquement une solution approchée d'un équation différentielle par la méthode d'Euler.

En fait, le module Python appelé `scipy.integrate` contient une fonction, appelée `odeint` (pour « ordinary differential équation integration »), qui permet de résoudre directement l'équation sans avoir à programmer soi-même la méthode de résolution. Les avantages sont que :

- on gagne du temps
- la résolution est en général meilleure qu'avec la méthode d'Euler « basique » car `odeint` utilise des schémas de résolution un peu plus sophistiqués que le schéma d'Euler explicite (même si les principes de base sont les mêmes).

On va rappeler brièvement la syntaxe de la fonction `odeint` sur un exemple simple d'équation du premier ordre :

$$\frac{dy}{dt} = -3y+5\mathrm{t}$$

que l'on voudrait résoudre sur l'intervalle de temps $[0\,s,\ 5\,s]$, avec comme condition initiale y(0) = 10.

```
import numpy as np                  # pour pouvoir travailler avec des tableaux
import matplotlib.pyplot as plt     # pour pouvoir tracer la solution
from scipy import integrate      # importation du module nécessaire à la résolution

def F(Y,t) : # définition de la fonction qui caractérise l'équation différentielle
     return -3*Y+5*t

dt = 0.001                          # choix du « pas » temporel
T = np.arange(0, 5, dt)             # création de la subdivision temporelle

Y0 = 10                          # condition initiale

Y = integrate.odeint(F, Y0, T)      # résolution de l'équation différentielle :
                                    # le tableau y renvoyé par odeint contiendra
                                    # les valeurs prises par la solution aux
                                    # différents instants de la subdivision

plt.plot(T,Y)           # tracé de la solution
```

Question : Taper ce code et observer la solution renvoyée. Vous paraît-elle cohérente ?

2) Cas d'une équation différentielle d'ordre 2 :

En mécanique, on rencontre très souvent des équations différentielles d'ordre 2 (tout simplement car le principe fondamental relie la force à l'accélération, qui est la dérivée seconde de la position). On va voir que la fonction `odeint` peut s'appliquer sans problème à une équation d'ordre 2 (en fait, elle peut être appliquée à une équation de n'importe quel ordre).

On a déjà vu qu'une équation différentielle d'ordre 2 peut être vue comme un système de deux équations différentielles d'ordre 1.

On va travailler sur l'équation d'ordre 2 suivante (correspondant à un oscillateur amorti forcé) :

$$\ddot{y} + 2\dot{y} + 36y = 40\cos(2t)$$

que l'on veut résoudre sur l'intervalle $[0\,s,\ 15\,s]$ avec comme conditions initiales $y(0) = 6$ et $\dot{y}(0) = 0$.

On remarque qu'en introduisant comme variable $v = \dot{y}$ l'équation différentielle du second ordre précédente peut être vue comme le système de deux équations du premier ordre :

$$\begin{cases} \dfrac{dy}{dt} = v \\ \dfrac{dv}{dt} = -2v - 36y + 40\cos(2t) \end{cases}$$

On peut aussi voir ce système comme une équation vectorielle. Si on introduit le vecteur $Y = \begin{pmatrix} y \\ v \end{pmatrix}$ (donc tel que Y[0] = y et Y[1] = v), l'équation s'écrit :

$$\frac{dY}{dt} = \begin{pmatrix} v \\ -2v - 36y + 40\cos(2t) \end{pmatrix} = \begin{pmatrix} Y[1] \\ -2\,Y[1] - 36\,Y[0] + 40\cos(2t) \end{pmatrix}$$

La résolution avec `odeint` se passe alors exactement comme à l'ordre 1 sauf que l'inconnue est maintenant un vecteur !

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate

def F(Y,t) :
      return np.array([Y[1],-2*Y[1]-36*Y[0]+40*np.cos(2*t)])

dt = 0.001
T = np.arange(0,15,dt)

Y0 = np.array([6,0])    # conditions initiales : c'est un vecteur contenant la
                        # position initiale et la vitesse initiale

Y = integrate.odeint(F,Y0,T)

plt.plot(T,Y)
```

Question : tapez ce programme et observez la solution obtenue. Vous paraît-elle cohérente au vu de l'équation ?

On remarque que Python renvoie deux courbes (l'une représentant la position y en fonction du temps et l'autre représentant la vitesse v en fonction du temps).

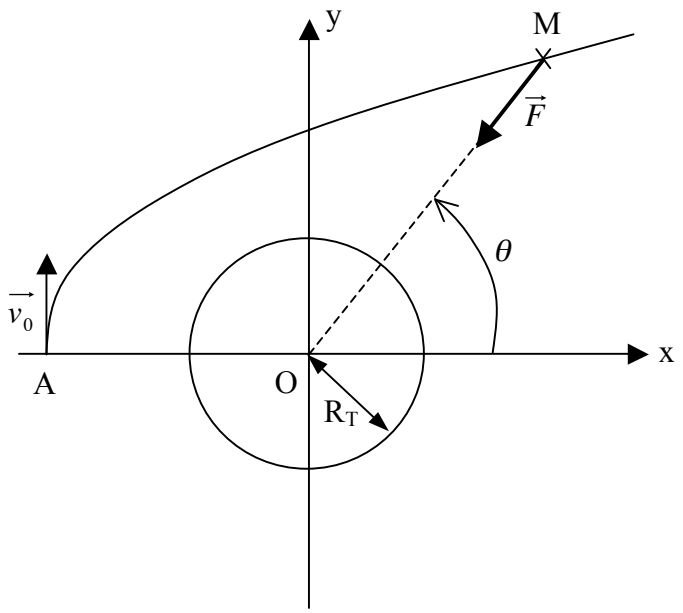
En effet, le résultat Y renvoyé par `odeint` est ici un tableau de deux colonnes, la première contenant toutes les valeurs de y (aux différents instants de la subdivision T) et la deuxième contenant toutes les valeurs de v.

Si on ne veut tracer que la position en fonction du temps, on écrira donc :

```
plt.plot(T, Y[ :,0])
```

de façon à ne sélectionner que la première colonne.

II Application à l'étude des trajectoire des satellites terrestres :



Données :

- *Masse de la Terre : $M_T = 5{,}97 \cdot 10^{24}$ kg*
- *Rayon de la Terre : $R_T = 6370$ km*
- *Constante de gravitation universelle : $G = 6{,}67 \cdot 10^{-11}$ u.s.i.*

1) Etude théorique :

On considère un satellite M de masse m initialement au point A de coordonnées x_A = - 8400 km (i.e. à 2000 km de la surface terrestre) et y_A = 0. On lui communique alors une vitesse initiale $\vec{v_0}$ dirigée selon $\vec{u_y}$. On se demande, en fonction de la valeur de v_o, quelle sera la nature de sa trajectoire. Pour chaque question, vous donnerez une expression littérale et une valeur numérique.

1) Pour quelle valeur $v_{o,c}$ de v_o le satellite entrera-t-il sur une orbite circulaire ?

2) Quelle est la valeur minimale $v_{o,min}$ de v_o pour que le satellite ne s'écrase pas sur la Terre ? (on négligera la présence de l'atmosphère terrestre).

2) Pour quelle valeur minimale $v_{o,lib}$ de v_o le satellite va-t-il pouvoir s'échapper de l'attraction terrestre ? (vitess de libération)

3) Quelle est la nature des trajectoires pour $v_{o,min} < v_o < v_{o,lib}$? Même question pour $v_o > v_{o,lib}$?

On se propose maintenant de vérifier ces résultats en résolvant numériquement (avec Python) les équations qui régissent le mouvement du satellite.

2) Résolution numérique avec Python :

1) Ecrire les équations différentielles satisfaite par x(t) et y(t) pour le satellite (les équations finales ne doivent pas faire intervenir l'angle θ).

2) Sous Python, écrire tout d'abord un programme permettant de tracer un cercle de rayon R_T (qui représentera la surface terrestre).
Indication : on pourra, par exemple, utiliser les équations paramétriques du cercle : $x = R_T \cos(\alpha)$ et $y = R_T \sin(\alpha)$ où α varie dans un intervalle de largeur 2π.

2) Ecrire ensuite un programme (utilisant la fonction `odeint`) permettant de résoudre les équations différentielles du mouvement, et donc de tracer la trajectoire du satellite (vous pourrez vous inspirez du

paragraphe I 2, en tenant compte du fait qu'il y a ici deux variables : x et y, au lieu d'une seule (y) dans l'exemple du paragraphe I 2).

3) Tracez sur une même figure différentes trajectoires possibles du satellite, en prenant différentes conditions initiales pour la vitesse :

a) $v_0 = v_{0,min}$
b) $v_0 = v_{0,c}$
c) $v_{0,c} < v_0 < v_{0,lib}$
d) $v_0 = v_{0,lib}$
e) $v_0 > v_{0,lib}$

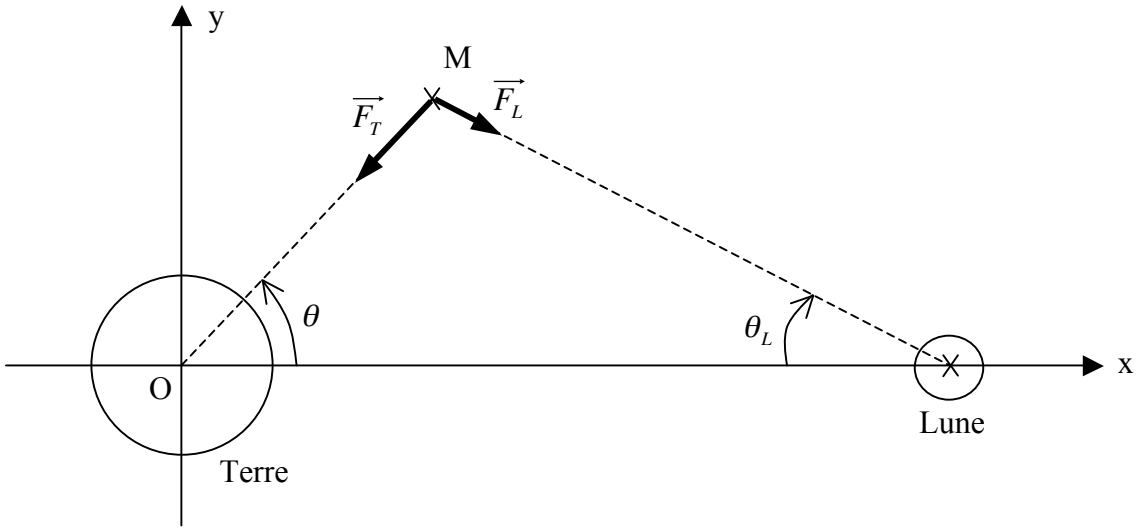
Dans chaque cas, les résultats de la simulation numérique sont-ils en accord avec vos prédictions ?

III Système Terre-Lune, trajectoire de retour libre et mission Apollo 13 :

On va maintenant rajouter la Lune dans notre étude et chercher des « trajectoires de retour libres » similaires à celle qui a permis aux astronautes de la mission Apollo 13 de rentrer sur Terre sains et saufs alors qu'ils n'avaient essentiellement plus de moteurs.

Données supplémentaires :

- *Rayon de la Lune : $R_L = 1737$ km*
- *Masse de la Lune : $M_L = 7{,}35 \cdot 10^{22}$ kg*
- *Distance Terre - Lune : D = 384 400 km*



1) Ecrire les nouvelles équations différentielles satisfaites par l'abscisse x(t) et l'ordonnée y(t) du vaisseau, en tenant compte maintenant de l'attraction terrestre et de l'attraction lunaire (dans un premier temps, on suppose que la Lune et la Terre sont toutes les deux fixes).

2) Modifier votre programme Python précédent pour résoudre ces équations différentielles. Rajouter également le tracé du cercle représentant la Lune.
On suppose que le vaisseau spatial se trouve initialement au point A de coordonnées (x_A = -16 400 km , y_A = 0), c'est à dire qu'il est sur l'axe des abscisses, à gauche de la terre, à une altitude de 10 000 km par rapport au sol (ce qui correspond à une orbite terrestre « moyenne », plus basse que l'orbite géostationnaire).
Quelle vitesse initiale v_0 (dirigée selon $\overrightarrow{u_y}$) faut-il lui communiquer afin qu'il entre sur une « trajectoire de retour libre » (voir page 2) qui lui permet de faire le tour de la Lune et de rentrer sur Terre en utilisant la force gravitationnelle de la Lune (idéalement, il faudrait qu'à l'arrivée le vaisseau « s'écrase » sur la surface terrestre).
Combien de temps le vaisseau spatial met-il pour parcourir sa trajectoire ?

3) Question plus difficile (seulement pour les élèves motivés !). La principale approximation faite ici était de considérer que la Lune est immobile, alors qu'en plusieurs jours, elle se déplace de façon considérable autour de la Terre. Sachant que la période de révolution de la lune autour de la Terre est T_L = 27 jours 7 heures 43 minutes, calculez la trajectoire réelle du vaisseau (qui tient compte du déplacement de la lune). Quelle est en fait la vitesse v_0 que l'on doit communiquer ?
Vous pouvez également (si vous êtes très fort en informatique) réaliser une animation où on voit la Lune et le vaisseau qui se déplacent (et le vaisseau qui arrive à contourner la Lune et à rentrer sur Terre, sans moteurs).