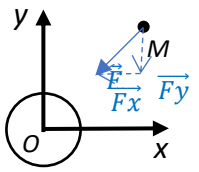
TP n° 21 : Simulation d'un mouvement à force centrale



On lance un satellite de masse $\boldsymbol{m = 1000\ kg}$ depuis la surface de la Terre dans l'objectif de le mettre en orbite. On considère qu'il n'y a pas de frottement et que seule la force gravitationnelle s'exerce sur ce satellite.
Notre objectif est de déterminer à l'aide d'une simulation numérique les trajectoires possibles pour ce satellite en fonction de sa vitesse initiale.

Vitesses cosmiques

On rappelle les expressions des deux vitesses cosmiques (démonstation dans le cours) :

La première vitesse cosmique (ou **vitesse de satellisation**) est la vitesse d'un objet satellisé en mouvement circulaire autour d'un astre de masse $\boldsymbol{m_a}$ à une altitude très petite devant son rayon $\boldsymbol{R_a}$:

$$\boldsymbol{v_{c_1} = \sqrt{\frac{Gm_a}{R_a}}}$$

La deuxième vitesse cosmique (ou **vitesse de libération)** est la vitesse minimale à communiquer à un objet situé initialement à la surface de la planète pour qu'il puisse échapper à l'attraction de la planète :

$$\boldsymbol{v_{c_2} = \sqrt{2\frac{Gm_a}{R_a}} = \sqrt{2} \times v_{c_1}}$$

Pour la terre avec $\boldsymbol{Mt = 5,972 \times 10^{24} kg}$ et $\boldsymbol{Rt = 6371\ km}$, $\boldsymbol{v_{c_1} \approx 8\ km.s^{-1}}$ et $\boldsymbol{v_{c_2} \approx 11\ km.s^{-1}}$.

Partie 1 : Détermination des équations du mouvement du satellite

A l'aide du schéma, appliquer le PFD au satellite situé au point **M** en coordonnées cartésiennes.

On exprimera les composantes de la force gravitationnelle $\boldsymbol{Fx}$ et $\boldsymbol{Fy}$ uniquement en fonction de $\boldsymbol{G}$, $\boldsymbol{m}$, $\boldsymbol{Mt}$, $\boldsymbol{x}$ et $\boldsymbol{y}$.

Partie 2 : Résolution numérique des équations du mouvement

Les équations du mouvement obtenues sont non-linéaires, c'est pourquoi on se propose de les résoudre numériquement avec Python.

1) Ouvrir Spyder, importer les bibliothèques suivantes.

```
from math import *
import numpy as np                      # pour le tracé des fonctions
import matplotlib.pyplot as plt         # pour le tracé des fonctions
from scipy.integrate import odeint      # pour la résolution des équadiff sur Python
```

2) Dans une première partie du programme , indiquer toutes les valeurs caractéristiques de la Terre : son rayon **Rt**, sa masse **Mt**, et la constante de gravitation universelle **G**.

3) Ensuite, indiquer les caractéristiques de l'objet lancé à la surface de la Terre : sa masse $\boldsymbol{m}$ et la vitesse initiale $\boldsymbol{v}$ à laquelle on souhaite le lancer. On pourra choisir une vitesse intermédiaire entre la vitesse de satellisation et la vitesse de libération.

4) Ecrire une fonction $\boldsymbol{trace\_cercle}(\boldsymbol{R})$ qui permet de tracer un cercle de rayon $\boldsymbol{R}$ en utilisant les coordonnées cartésiennes.

On commencera par créer un tableau de 100 valeurs d'angle ***theta*** *variant entre* $\mathbf{0}$ *et* $\mathbf{2}\boldsymbol{\pi}$ *avec la fonction* $\boldsymbol{linspace()}$.

Appeler cette fonction pour tracer un cercle correspondant à la Terre.

5) Ecrire une fonction $\boldsymbol{Fg}(\boldsymbol{x},\boldsymbol{y})$ renvoyant les composantes $\boldsymbol{Fx}$ et $\boldsymbol{Fy}$ respectivement selon $\overrightarrow{\boldsymbol{U_x}}$ et $\overrightarrow{\boldsymbol{U_y}}$ de la force de gravitation exercée par la Terre sur le satellite situé au point $\boldsymbol{M}$ de coordonnées $(\boldsymbol{x},\boldsymbol{y})$.

La fonction $\boldsymbol{Odeint}$ de *scipy.integrate ne permet pas d'intégrer des équations différentielles d'ordre 2, il faut donc écrire les équations du mouvements sous la forme d'un système d'équations du premier ordre.*

6) Recopier et compléter la fonction $\boldsymbol{dynamique}(\boldsymbol{S},\boldsymbol{t})$ renvoyant un tableau contenant les composantes dérivées du vecteur état $\boldsymbol{S} = (\boldsymbol{x},\boldsymbol{y},\boldsymbol{vx},\boldsymbol{vy})$.

```
def dynamique(S, t):
    x, y, vx, vy = S
   Fx,Fy = Fg(x, y)
   …. A compléter
    return [vx, vy, ax, ay]
```

7) Recopier la fonction $\boldsymbol{simulation}(\boldsymbol{v0})$ qui permet de résoudre le système d'équations différentielles numériquement.

```
def simulation(v0):
    S0 = [Rt, 0, 0, v0]      # Conditions initiales
    t = np.linspace(0, 20000, 5000)      # Temps : quelques heures
    sol = odeint(dynamique, S0, t)
    x = sol[:,0]
    y = sol[:,1]
    return x, y
```

8) Appeler la fonction $\boldsymbol{simulation}(\boldsymbol{v0})$**.** Recopier le code suivant pour tracer la trajectoire obtenue avec odeint, qui s'ajoute alors au tracé de la Terre réalisé à la question 3.

```
plt.figure(figsize=(6,6))
plt.plot(x, y, 'r', label="Trajectoire")
plt.axis('equal')
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.title(f"Trajectoire pour v0 = {v0/1000} km/s")
plt.legend()
plt.grid()
```

9) Recopier puis completer le code suivant à la suite de votre programme pour tracer l'énergie mécanique du satellite et l'énergie potentielle effective.

 # Energie mécanique et énergie potentielle effective

*r = np.linspace(0.1*Rt, 100*Rt, 2000)*

Eeff = … à compléter
Em = … à compléter

plt.figure(figsize=(7,5))
plt.plot(r/Rt, Eeff, label="Potentiel effectif")
plt.axhline(Em, linestyle='--', label="Énergie mécanique")
plt.xlabel("r / Rt")
plt.ylabel("Énergie (J)")
plt.title("Potentiel effectif")
plt.legend()
plt.grid()
*plt.ylim(min(Eeff)*1.2, abs(Em)*1.5)*

plt.show()

10) Tester le programme avec les conditions initiales $(\boldsymbol{Rt}, \mathbf{0}, \mathbf{0}, \boldsymbol{v0})$ avec $\boldsymbol{v_0} = [\mathbf{8}\,;\mathbf{10}\,;\mathbf{12}]\ \boldsymbol{km.s^{-1}}$.

Conclure.