Le but de ce TP est d'apprendre à utiliser la librairie SciPy.

Dans les modules scientifiques de Python, il existe une collection impressionnante de fonctions réalisant des opérations diverses sur les tableaux numériques de NumPy. Il faut surtout les utiliser, plutôt que de réinventer la roue en recodant (probablement moins bien) une fonctionnalité qui existe déjà par ailleurs !

Dans NumPy, on a déjà vu qu'il existe beaucoup d'opérations permettant par exemple de
- générer des tableaux particuliers : np.arange, np.ones, np.linspace, ...
- faire des opérations à partir des valeurs du tableau : np.sum, np.sin, np.histogram, etc.
- changer l'agencement des valeurs d'un tableau, sa forme, ou encore créer un nouveau tableau à partir de plusieurs autres : np.reshape, np.concatenate.
- etc. ...

Le module SciPy est la boîte à outils numérique pour les tableaux NumPy. On trouve dans SciPy les opérations de manipulation / traitement de données numériques classiques, mais spécifiques à un type d'application (algébre linéaire, statistiques, etc.). Ce sont donc des fonctions plus "haut niveau" que celles de NumPy.

SciPy est un module stable, bien testé et relativement bien documenté.

`http://docs.scipy.org/doc/`, `http://docs.scipy.org/doc/scipy/reference/`

```
>>> import scipy
```

Le module SciPy réalise les différentes opérations sur des tableaux numériques (`ndarray`) de NumPy. On peut donc directement utiliser ces tableaux comme arguments pour les différentes fonctions

```
>>> from scipy import linalg
>>> mat = np.array([[1, 2], [2, 4]])
>>> mat
array([[1, 2],
    [2, 4]])
>>> linalg.det(mat)
0.0
```

1 Pendule simple

Pour montrer l'utilisation de SciPy, nous allons nous intéresser à l'intégration d'équations différentielles, en considérant des systèmes dynamiques à base de pendules mécaniques.

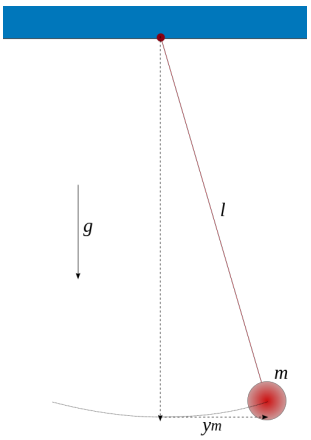

FIGURE 1 : Pendule simple

En écrivant la relation fondamentale de la dynamique (conservation de la quantité de mouvement), on obtient

l'équation du pendule simple donnée par

$$\ddot{\theta} + \omega^2 \sin\theta = 0$$

où θ est l'angle du pendule par rapport à la verticale, et on note avec un point la dérivée temporelle.

Pour les petites oscillations on peut faire l'approximation $\sin\theta \approx \theta$. Quand l'approximation n'est pas valide il faut intégrer numériquement cette équation différentielle pour obtenir l'évolution de la position et de la vitesse angulaire du pendule, au cours du temps.

Il nous faut donc disposer d'un intégrateur d'équations différentielles, que l'on peut s'attendre à trouver dans SciPy. Mais quelle est la fonction correspondante ? On peut consulter le sommaire de l'aide `http://docs.scipy.org/doc/scipy/reference/index.html`. Il existe un sous-module integrate, qui contient lui-même une fonction odeint

```
from scipy.integrate import odeint
```

Regarder la doc de la fonction `http://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html#scipy.integrate.odeint` et l'exemple `http://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html#ordinary-differential-equations-odeint`.

Pour commencer, il faut mettre l'équation différentielle du 2nd ordre sous la forme d'un système d'équations du premier ordre

```
def simple_pendulum(theta_thetadot, t):
    theta, theta_dot = theta_thetadot
    return [theta_dot, - np.sin(theta)]
```

correspondant à

$$\frac{d\theta}{dt} = \dot{\theta},$$
$$\frac{d\dot{\theta}}{dt} = -\sin\theta.$$

Nous pouvons maintenant intégrer une trajectoire à partir d'une condition initiale

```
>>> t = np.linspace(0, 5 * np.pi, 1000)
>>> sol = odeint(simple_pendulum, (np.pi/3, 0), t)
```

Nous pouvons par exemple vérifier la conservation de l'énergie mécanique au cours du temps

```
import matplotlib.pyplot as plt
from scipy.integrate import odeint

def simple_pendulum(theta_thetadot, t):
    theta, theta_dot = theta_thetadot
    return [theta_dot, - np.sin(theta)]

theta_thetadot = (np.pi / 3, 0)
t = np.linspace(0, 5 * np.pi, 1000)
sol = odeint(simple_pendulum, (np.pi/3, 0), t)
theta, theta_dot = sol.T

E_kin = 1./ 2 * theta_dot ** 2
E_pot = 1 - np.cos(theta)
E_mech = E_kin + E_pot

plt.figure()
ax = plt.gca()
plt.plot(t, E_kin, 'o-', lw=2, label=u'$E_\mathrm{kin}$')
plt.plot(t, E_pot, 'o-', lw=2, label=u'$E_\mathrm{pot}$')
plt.plot(t, E_mech, lw=2, label=u'$E_\mathrm{mech}$')
plt.xticks(np.pi * np.arange(5), [0, u'$\pi$', u'$2\pi$', u'$3\pi$', u'$4\pi$'])
plt.xlabel(u'$t$', fontsize=26)
plt.title(u'Conservation de l\'energie')
plt.gca().tick_params(axis='both', which='major', labelsize=20)
plt.legend(loc='best')
plt.grid()
plt.show()
```

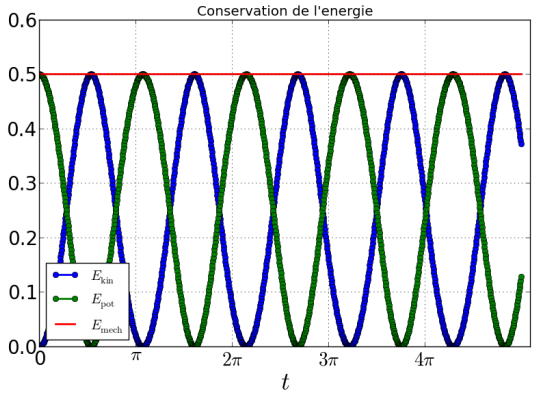

FIGURE 2 : Conservation de l'énergie

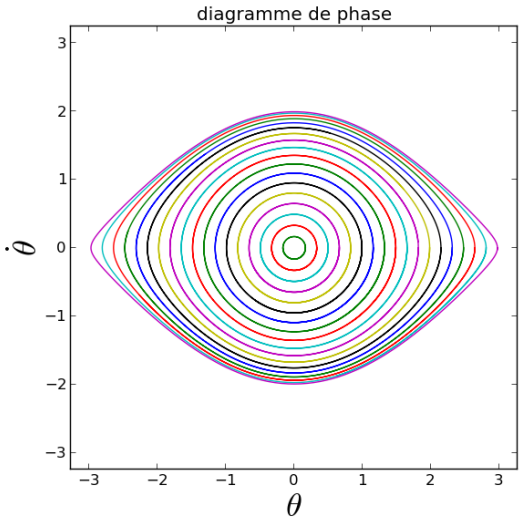

FIGURE 3 : Diagramme de phase

2 Exercice

Ecrire un script python pour construire le diagramme des phases du pendule simple, représenté Fig. 3. Pour celà, il faut

- intégrer l'équation différentielle pour différentes conditions initiales entre 0 et π.
- représenter $\dot{\theta}$ en fonction de θ pour les différentes solutions.