Odeint et le pendule simple

Le pendule simple a pour équation (voir TD de dynamique) y''

On ne sait pas résoudre cette équation différentielle non linéaire, python va le faire pour nous.

Dans tout le TP, on lâchera le pendule à vitesse nulle : à t=0, y=0

1. Dans quelle approximation peut on se placer pour résoudre une équation approchée ? Le faire, calculer à la main la période de la solution.
2. On veut étudier la différence entre les solutions de l'équation linéarisée et celles de l'équation initiale. Nous nous interesserons à la période des oscillations (notée ici Pe, pour ne pas confondre avec le tableau temporel T). Pour mettre facilement en évidence une différence, comment choisir \omega pour avoir une période linéarisée de 1 seconde ? On gardera cette valeur de \omega pendant toute l'activité.

De même, dans toute l'activité, on lâchera le pendule à vitesse nulle : à t=0, y=0

3. Compléter le script suivant avec odeint pour calculer et tracer la solution de l'équation non linéarisée, avec la condition initiale en angle suivante : y(0)=10° (à convertir en radians !)

On superposera sur le graphe la solution de l'équation linéarisée pour faire apparaitre une éventuelle différence

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as np
plt.clf() #efface la figure précédente

# fonction de l'equation differentielle
def F(Y,t):
    y = Y[0]
    yp = Y[1]

# fabrication de la sortie : y' est déja là, c'est yp !!!
# ne reste qu'a exprimer y'' avec l'equation differentielle
    ypp=-np.sin(y)*4*3.1415927**2
    return (yp,ypp)

# intervalle de resolution, on prend 1000 valeurs de t entre 0 et 1.1 s
T = np.linspace (0 ,1.1 ,10000)

# Conditions initiales
CI_y=60*3.14159/180  # CI sur y
CI_yp=0 # ci SUR y'
CI=(CI_y,CI_yp)

Sol = odeint (F,CI,T)
Y_sol=Sol[:,0]
plt.plot(T,Y_sol)
Y_ref=CI_y*np.cos(2*3.14159*T)
plt.plot(T,Y_ref)
plt.show()
```

3. Refaire la même chose en modifiant le script ci-dessus, pour des angles initiaux de 30 et 60° (pendule toujours lâché sans vitesse initiale). Que constate-t-on ?

La période est donc fonction de l'angle initial T=f(y_0). Nous allons tracer cette courbe.

4. Compléter la fonction \pyth|Periode(Y,t)| qui prend comme argument un tableau \pyth|Y| (à une dimension) et le tableau \pyth|t | des temps associées, et retourne la période de Y. On peut remarquer que T/4 est le temps que met y à passer la première fois par zero.

```
def Periode(Y,t):
    N = len(Y)
    for i in range(N):
        if Y[i]<0:
            return t[i]*4
    print ('pas de période trouvée')
    return 0
```

5. Compléter le script suivant, de manière à tracer T=f(y_0) pour les angles initiaux répartis tous les 5° entre 0 et 90°.

```
angles_degres = np.linspace (1,150,20)
print (angles_degres)
angles_radians = angles_degres*np.pi/180

Periodes=[]
for angle in angles_radians:
     CI = (angle,0)
     Sol = odeint (F,CI,T)
     Y_sol=Sol[:,0]
     per = Periode(Y_sol,T)
     print (per)
     Periodes.append(per)

plt.clf()
plt.scatter(angles_degres,Periodes)
plt.show()
```

```
[  1.           8.84210526  16.68421053  24.52631579  32.36842105
  40.21052632  48.05263158  55.89473684  63.73684211  71.57894737
  79.42105263  87.26315789  95.10526316 102.94736842 110.78947368
 118.63157895 126.47368421 134.31578947 142.15789474 150.        ]
1.0002200220022004
1.0015401540154016
1.0055005500550056
1.0116611661166117
1.0204620462046206
1.031903190319032
1.0459845984598461
1.0631463146314633
1.0833883388338834
1.1075907590759078
1.1353135313531355
1.167876787678768
1.2057205720572057
1.24972497249725
1.3012101210121014
1.3614961496149616
1.4332233223322333
1.5199119911991201
1.6264026402640266
1.7623762376237626
```