```python
# -*- coding: utf-8 -*-
"""
Created on Sun Mar  2 18:01:22 2014

@author: dconduche
"""

"""Constantes physiques (ex 5), donc en variables globales"""

import numpy as np
import matplotlib.pyplot as plt

g=9.81   #pesanteur
L=.1     #longueur caractéristique
omega0=np.sqrt(g/L) #pulsation propre, pour la comparaison avec le cos


"""Cette fonction est à connaitre et comprendre"""
def Euler(F,y0,a,b,n):
    x=np.linspace(a,b,n) #création de la subdivision régulière de [a,b], n points. x[0]=a et x[n-1]=b (dernier point)
    y=np.empty((n,np.size(y0))).squeeze()
      #création du tableau qui va contenir y[k]
      #on aurait pu se contenter de y=np.empty(n) mais ça ne traite pas le cas où y est un vecteur
      #explication détaillée :
          #np.size(y0) renvoie la taille de y0. 1 si c'est un nombre (ex:float, int) et un entier k si c'est un tableau contenant k cases.
          #on crée donc un tableau de taille n x k, et .squeeze() permet de le transformer en un tableau de taille n si k=1.
      #(comprendre le .squeeze() n'est pas très important)
    pas=(b-a)/float(n-1)
    y[0]=y0
    for k in range(n-1):
        y[k+1]=y[k] + F(x[k],y[k])*pas
      #On retourne les valeurs de y calculées pour la subdivision régulière en n points sur [a,b]:
    return y
"""fin de fonction à connaitre et comprendre"""


def Init_Affichage():
    """Initialise l'affichage dans un mode 'cours de maths'"""
    ax = plt.gca()
     #Axes horizontaux.
    ax.spines['top'].set_color('none')
    ax.xaxis.set_ticks_position('bottom')
```

```
    ax.spines['bottom'].set_position(('data',0))
     #Axes verticaux.
    ax.spines['right'].set_color('none')
    ax.yaxis.set_ticks_position('left')
    ax.spines['left'].set_position(('data',0))

def Affichage_1d(y,f,a,b):
    """tracé de fonction y=f(x) : affichage"""
      #y contient les ordonnées des points à afficher,
      #f une fonction usuelle (pour comparer),
      #on trace sur [a,b]
    n=np.size(y) #nombre de points à afficher
    x=np.linspace(a,b,n) #subdivision (cf Euler pour explication détaillée)
    Init_Affichage()
     #label : donne un nom aux graphes
    plt.plot(x,map(f,x), c='red',label='fonction')
    plt.plot(x,y, c='black',label='Euler n='+str(n))
     #affichage de la légende (les noms des graphes)
    plt.legend(loc=9)

def Portrait_phase(y,f,fprime,a,b,legende=True):
    """portrait de phase : affichage"""
      #y contient un vecteur [y[0],y[1]] à afficher dans le plan
      # (f,fprime) contient les fonctions solutions théoriques (ou approchées théoriques, pour le pendule)
      #legende : affichage ou non de la legende (booléen, par défaut : True)
    n=len(y) #nombre de points à afficher
    Init_Affichage()
    plt.plot(y[:,0],y[:,1], c='black',label='Euler n='+str(n))
    x=np.linspace(a,b,n) #subdivision (cf Euler pour explication détaillée)
    plt.plot(map(f,x),map(fprime,x), c='red',label='fonction')
     #affichage de la légende (les noms des graphes)
    if legende :
        plt.legend(loc=1)


"""Rappel :
        lambda arguments: expression
   est synonyme de :
        def <lambda>(arguments):
            return expression
"""
```

```python
def Exercice4(n):
    a=0
    b=3
    ExpEu=Euler(lambda x,y : y,1,a,b,n)
    Affichage_1d(ExpEu,np.exp,a,b)

def F_pendule_simple(t,y):
      # Equa diff du pendule non amorti
      #Ici y est un vecteur, deux coordonnées : [y[0],y[1]]
    return np.array([y[1],-g/L*np.sin(y[0])])


def PenduleSimple(n):
    a=0
    b=3
    y0=[np.pi/2,0]
    PenEu=Euler(F_pendule_simple,y0,a,b,n) #Resultat de méthode d'Euleur pour le pendule
#    print(np.size(PenEu[:,0])) #pour le débug.
    y=PenEu*np.array([1/y0[0],1/(y0[0]*omega0)])
     #On divise le coeff PenEu[0] par y0[0]; et PenEu[1] par y0[0]*omega0
    f_theorique_approchee=lambda x : np.cos(omega0*x)
    f_theorique_approchee_derivee=lambda x : -np.sin(omega0*x)
    """La trajectoire theta en fonction du temps"""
    Affichage_1d(y[:,0],f_theorique_approchee,a,b)
    plt.show()
    plt.clf()
    """Le portrait de phase de la trajectoire précédente"""
    Portrait_phase(y,f_theorique_approchee,f_theorique_approchee_derivee,a,b)
    plt.show()
    plt.clf()
    """Avec plusieurs angles theta0 de départ"""
    nombre_angles=10
    for theta0 in np.linspace(np.pi/nombre_angles,np.pi,nombre_angles):
        #On coupe l'intervalle [0,pi] en 11. En enlevant theta0=0
        y=Euler(F_pendule_simple,[theta0,0],a,b,n)*np.array([1/theta0,1/(theta0*omega0)])
        Portrait_phase(y,f_theorique_approchee,f_theorique_approchee_derivee,a,b,legende=False)
    plt.show()


"""Exercice 4 : paramètre n = nombre d'éléments dans la subdivision
     test de la méthode d'Euler pour y'=y,
     comparaison graphique avec la solution théorique (exp)"""
```

```
Exercice4(10**2)
plt.show()
plt.clf()
Exercice4(10)
plt.show()
plt.clf()


""" Sur [0,3]:
   Avec n=10, la méthode d'Euler donne un résultat loin de la réalité (même pas un semblant de périodicité).
   Avec n=10**3 le résultat est plus probant, mais diverge (absurde physiquement).
   Avec n=10**5 le résultat (à l'oeil nu) semble satisfaisant."""

PenduleSimple(10**4)
```