Introduction

On souhaite tracer la solution de l'équation différentielle suivante:

$$\frac{d^2x}{dt^2}(t) + \omega_0(1-r)\frac{dx}{dt}(t) + \omega_0^2 x(t) = 0$$

Question 1

En introduisant une variable $v(t) = \frac{dx}{dt}(t)$, réécrire l'équation différentielle du second ordre sous la forme d'un système de deux équations couplées du premier ordre:

$$\begin{cases} \frac{dx}{dt}(t) &= v(t) \\ \frac{dv}{dt}(t) &= -\omega_0(1-r)v(t) - \omega_0^2 x(t) \end{cases}$$

Question 2

Ecrire la fonction python correspondante f(t,S,w0,r) où:

- t est le temps,
- S = [x,v],
- omega0 et r sont les deux paramètres de l'équation différentielle.

In [2]:
```
def f(t,S,w0,r):
    dx_dt = S[1]
    dv_dt = -w0*(1-r)*S[1]-w0**2*S[0]
    return [dx_dt,dv_dt]
```

Question 3

Pour résoudre l'équation différentielle, on utilisera la fonction solve_ivp du module integrate de scipy:

solve_ivp(fun, t_span, y0, method='RK45', args=None)

Les arguments de cette fonction sont:

- fun: la fonction qui définit l'équation différentielle à étudier.
- t_span: Intervalle d'intégration sous la forme [t0,tf].
- y0: les conditions initiales sous forme d'une liste [x0,v0].
- method: la méthode d'intégration. Ici, on se limitera à utiliser la méthode RK45
- args: les arguments éventuels de la fonction fun.

La fonction retourne un dictionnaire dont l'une des clés est la solution.

On pourra trouver des exemples d'utilisation au lien suivant:

https://cpge.frama.io/fiches-cpge/Python/%C3%89quation%20diff%C3%A9rentielle/0-Equation%20diff%C3%A9rentielle/

Remplir les arguments de la fonction ci-dessous afin d'obtenir la solution de l'équation différentielle avec $x_0 = 0.1, v_0 = 0, T_0 = \frac{2\pi}{\omega_0}, r = 0.2$

In [3]:
```
from scipy.integrate import solve_ivp
T0 = 1
tf = 4*T0 #on prend un temps d'intégration égal à 4*T0
w0 = 2*3.1415/T0
r = 0.2
solution = solve_ivp(f,[0,tf],[0.1,0],method='RK45',args=[w0,r])
```

Question 4

Déterminer la liste des clés du dictionnaire solution. Quel est le type de la clé 't'? de la clé 'y'?

In [4]:
```
solution.keys()
```

Out[4]:
```
dict_keys(['t', 'y', 'sol', 't_events', 'y_events', 'nfev', 'njev', 'nlu', 'status', 'message', 'success'])
```

In [5]:
```
type(solution['t'])
import numpy
numpy.shape(solution['t'])
```

Out[5]:
```
(28,)
```

In [6]:
```
type(solution['y'])
numpy.shape(solution['y'])
```

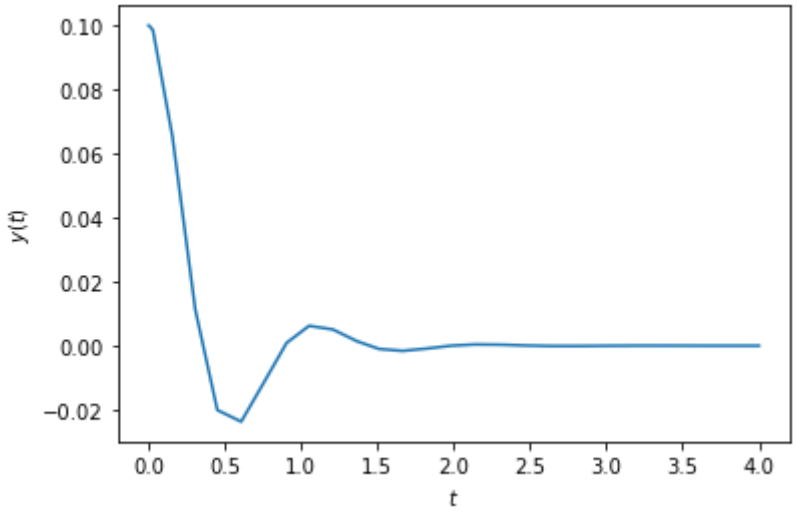
Out[6]:
```
(2, 28)
```

La clé 't' est un tableau numpy contenant les valeurs du temps dans l'intervalle [0,tf]. La clé 'y' est un tableau numpy contenant les valeurs de la position et de la vitesse pour chaque valeur de t.

Question 5

Tracer l'allure de $x(t)$. A-t-on un oscillateur?

In [7]:
```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(solution['t'],solution['y'][0])
plt.xlabel('$t$') #les dièses permettent de mettre le nom des axes en italique
plt.ylabel('$y(t)$')
plt.show()
```

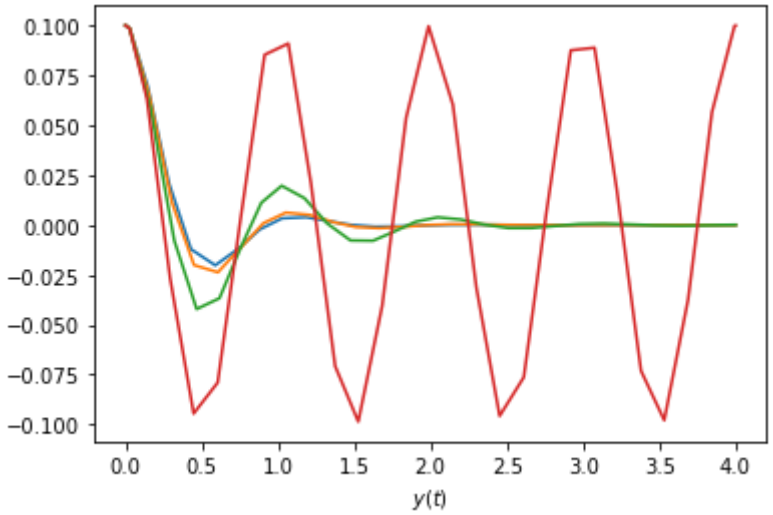


On trouve une courbe typique d'un oscillateur amorti. Pour r = 0.2, on a Q = 1/0.8 = 1.25 > 0.5. Le régime est pseudo-périodique.

Question 6

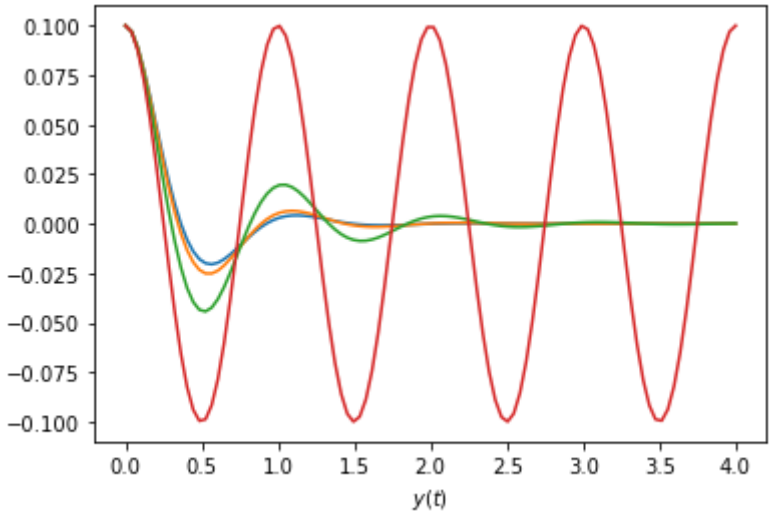
A l'aide d'une boucle for, tracer sur un même graphe l'évolution de $x(t)$ pour $r = 0.1, r = 0.2, r = 0.5, r = 1.0$. Que retrouvez-vous? Tracer ensuite l'évolution de $x(t)$ pour $r = 1.2, r = 2.0$. Que remarquez-vous?

In [8]:
```
plt.figure()
for r in [0.1,0.2,0.5,1.0]:
    solution = solve_ivp(f,[0,tf],[0.1,0],method='RK45',args=[w0,r])
    plt.plot(solution['t'],solution['y'][0])

plt.xlabel('$y$')
plt.xlabel('$y(t)$')
plt.show()
```

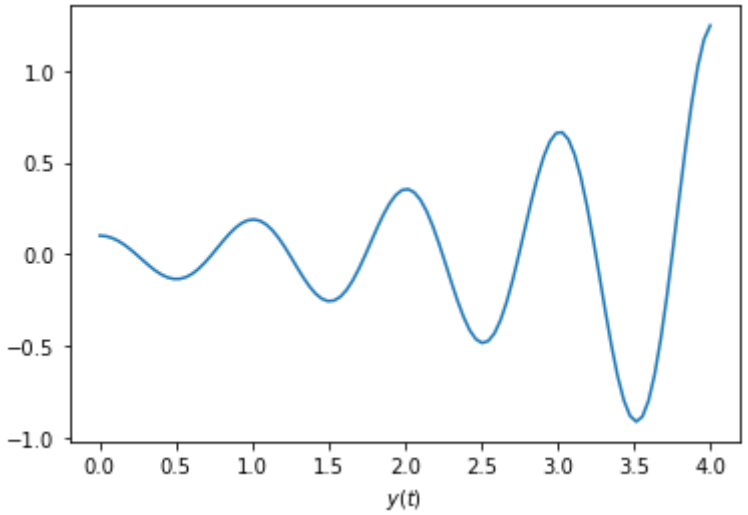


Plus r augmente, plus le régime est amorti. Pour $r = 1$, on retrouve les oscillations d'un oscillateur harmonique. On s'aperçoit néanmoins que le signal est sous-échantillonné (manque de points de mesure qui rend la courbe peu lisse). on peut améliorer la forme de cette courbe en précisant le nombre de points de mesures dans la fonction solve_ivp.

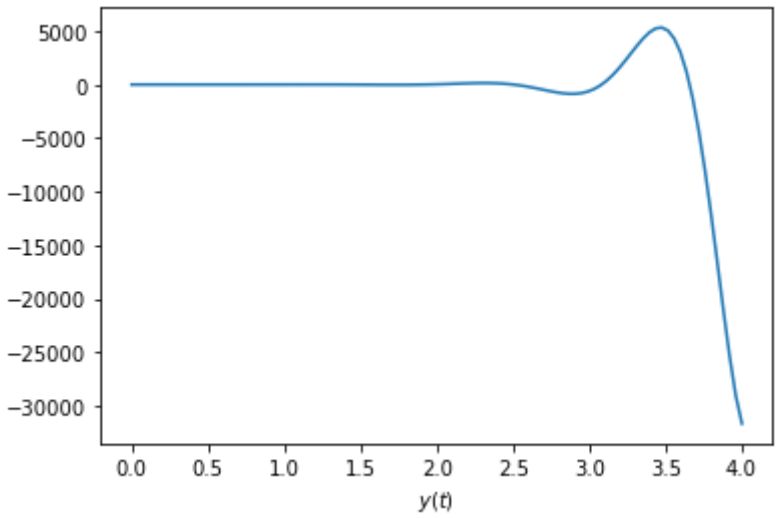
In [9]:
```
plt.figure()

#Pour prendre plus de points de mesures, on utilise la fonction linspace du modu
for r in [0.1,0.2,0.5,1.0]:
    solution = solve_ivp(f,[0,tf],[0.1,0],t_eval= numpy.linspace(0,tf,100),metho
    plt.plot(solution['t'],solution['y'][0])

plt.xlabel('$y$')
plt.xlabel('$y(t)$')
plt.show()
```



In [10]:
```
plt.figure()

r = 1.2
solution = solve_ivp(f,[0,tf],[0.1,0],t_eval= numpy.linspace(0,tf,100),method='R
plt.plot(solution['t'],solution['y'][0])

plt.xlabel('$y$')
plt.xlabel('$y(t)$')
plt.show()
```



In [11]:
```
plt.figure()

r = 2.0
solution = solve_ivp(f,[0,tf],[0.1,0],t_eval= numpy.linspace(0,tf,100),method='R
plt.plot(solution['t'],solution['y'][0])

plt.xlabel('$y$')
plt.xlabel('$y(t)$')
plt.show()
```



Pour $r > 1.0$, la solution diverge. Plus r augmente, plus la solution diverge violemment.

In [ ]: