Résolution d'équations différentielles avec Python

I. Méthode d'Euler

On considère une équation différentielle d'ordre 1 avec condition initiale $\begin{cases} y' = F(y,t) \\ y(t_0) = y_0 \end{cases}$.

Exercice d'application 1. On considère l'équation différentielle $y' + y = t + 1$. Définir en Python une fonction F de deux variables telle que $y' = F(y,t)$.

Revenons à la résolution. Le but est de calculer des valeurs approchées de y (notre fonction inconnue qui dépend du temps t) pour $t \geq t_0$. *On prend souvent* $t_0 = 0$ *pour simplifier les calculs.*

On commence par discrétiser notre intervalle $[t_0, t]$. Pour cela, on utilise les fonctions **arange** ou **linspace** du module **numpy** (dépendant de si l'on veut choisir le pas de discrétisation h ou si l'on veut choisir le nombre de points entre t_0 et t. Par exemple, les instructions suivantes permettent de construire le tableau $T = T_2 = [0, 0.01, 0.02, \ldots, 2]$:

```
import numpy as np
T=np.arange(0,2.01,0.01)
T2=np.linspace(0,2,201)
```

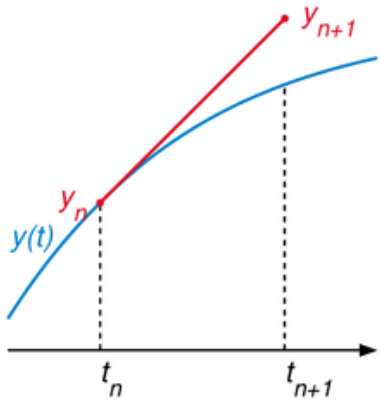
Exercice d'application 2. Définir le tableau $T_3 = [0, 0.2, 0.4, 0.6, \ldots, 2]$.

Si on note $T = [t_0, t_1, \ldots, t_N]$, on va alors calculer par récurrence les valeurs de $y'(t_k)$ et $y(t_k)$ pour $k \in [\![0, N]\!]$ en utilisant l'équation différentielle vérifiée par y. Ainsi :

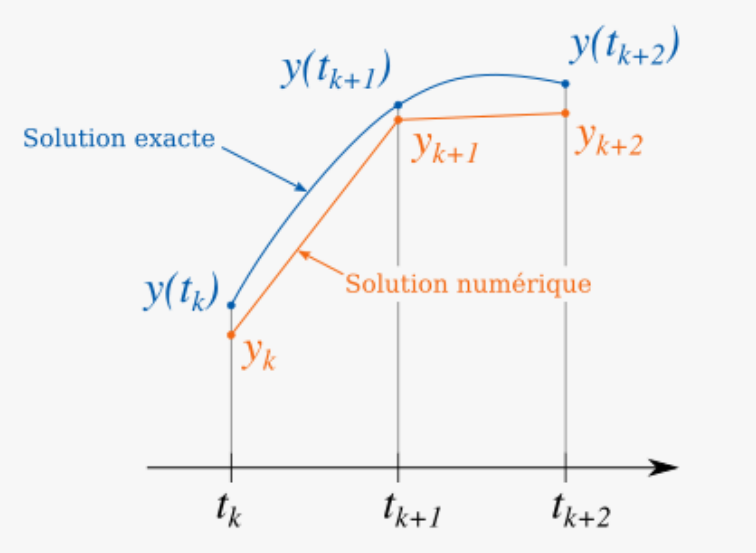
- $y(t_0) = y_0$ et $y'(t_0) = F(y(t_0), t_0)$.
- On pose alors, en approchant la fonction y en t_0 par sa tangente : $y(t_1) = y(t_0) + (t_1 - t_0)y'(t_0)$. On calcule ensuite $y'(t_1) = F(y(t_1), t_1)$.
- De manière générale, pour $k \in [\![0, N-1]\!]$, on pose

$$\begin{cases} y(t_{k+1}) = y(t_k) + (t_{k+1} - t_k)y'(t_k) \\ y'(t_{k+1}) = F(y(t_{k+1}), t_{k+1}) \end{cases}.$$

L'idée de cet algorithme est d'approcher la fonction (inconnue) par sa tangente que l'on calcule de manière successive :



Petite précision : ce que l'on a noté $y(t_k)$ ou $y'(t_k)$ précédemment ne sont pas les valeurs exactes de $y(t_k)$ et de $y'(t_k)$ mais devrait plutôt être noté différemment (par exemple y_k et y'_k). L'idée ici est donc bien de calculer $[y_0, y_1, \ldots, y_n]$ et c'est la valeur de y_n qui nous intéresse. Une illustration pour mieux illustrer ceci :

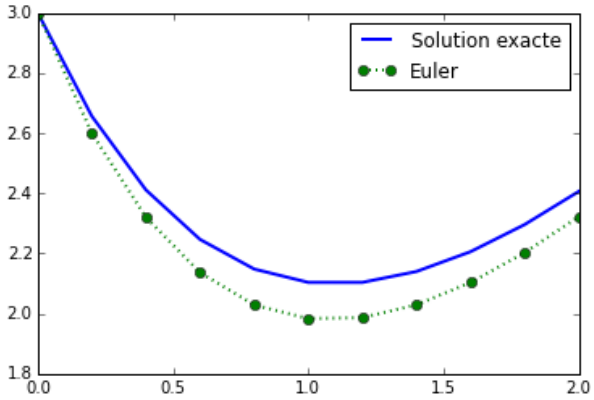


Ainsi pour coder la méthode d'Euler, notre script peut ressembler à (en supposant que la fonction F est bien définie et le tableau T également et que numpy est importé sous le nom np) :

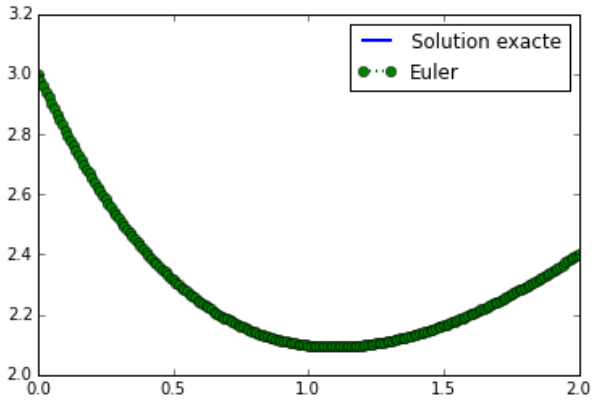
```
def Euler(y0,T):
    n=len(T)
    Y=np.zeros(n)
    Y1=np.zeros(n)
    Y[0]=y0
    Y1[0]=F(y0,T[0])
    for k in range(n-1):
        Y[k+1]=Y[k]+(T[k+1]-T[k])*Y1[k]
        Y1[k+1]=F(Y[k+1],T[k+1])
    return(Y)
```

On peut alors effectuer le tracé. Voici par exemple le tracé de l'équation $y' + y = t + 1$ sur $[0, 2]$ en prenant $y_0 = 3$ et $t_0 = 0$ et le tableau T_3 comme discrétisation. *On a pris une équation différentielle linéaire d'ordre 1 que l'on sait résoudre, ici la solution est* $y(t) = y_0 e^{-t} + t$.

```
import matplotlib.pyplot as plt

sol=Euler(3,T3)
plt.plot(T3,3*np.exp(-T3)+T3,linewidth=2,label='Solution exacte')
plt.plot(T3,sol,linestyle='dotted',marker='o',linewidth=2,label='Euler')
plt.legend()
plt.show()
```



Le pas de discrétisation est ici très grand d'où le grand écart entre la solution réelle et la solution approchée. Recommencer l'exemple précédent avec le tableau T qui a un pas de discrétisation de 0.01. On obtient normalement le tracé suivant, où la solution exacte et l'approchée se superposent :



Exercice d'application 3. Appliquer la méthode d'Euler à la résolution de l'équation différentielle $y' = y + t + 1$ (où la solution théorique est cette fois $y(t) = y_0 e^t + t$) en prenant $y_0 = 3$, $t_0 = 0$ et tracer les deux courbes sur $[0, 3]$ en prenant un pas de discrétisation assez petit. Que remarque-t-on ?

Exercice d'application 4. Appliquer la méthode d'Euler à la résolution de l'équation différentielle $y' = y^2 - 2\cos(t)$ et $y(0) = 1$ sur $[0, 3]$ et tracer la solution approchée.

II. Résolutions avec Odeint

Cette partie pourra vous être utile pour vos TIPEs quand vous rencontrerez des équations différentielles que l'on ne sait pas résoudre théoriquement. Des fonctions sont déjà implémentées dans Python pour résoudre les équations différentielles sans avoir à recoder la méthode d'Euler. Nous allons voir la syntaxe permettant d'utiliser la fonction **Odeint**.

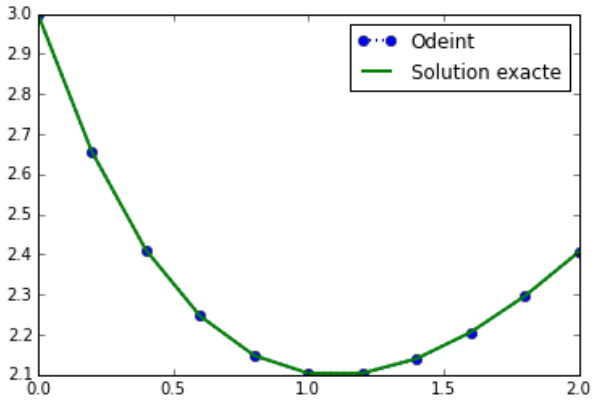
II.1. Équations d'ordre 1

La première étape est comme précédemment de mettre l'équation différentielles sous la forme $y' = F(y, t)$. Il suffit alors de définir la fonction F (qui dépend toujours de deux variables, y et y) et d'utiliser la fonction **odeint** (qu'il faut importer au préalable). Cette fonction prend en paramètre la fonction F, une condition initiale y_0 (qui sera la valeur de y pour la première valeur de t) et un tableau de la forme $[t_0, t_1, \ldots, t_n]$ donnant le pas de la discrétisation.

En sortie, la fonction **odeint** renvoie le tableau $[y(t_0), y(t_1), \ldots, y(t_n)]$ contenant les valeurs approchées de la solution de notre équation différentielle.

Reprenons l'équation $y' + y = t + 1$ avec la condition initiale $y(0) = 3$ que l'on sait résoudre explicitement (la solution est $y(t) = 3e^{-t} + t$). On a alors $y' = -y + t + 1$ donc on peut résoudre avec odeint sur $[0, 2]$ de la manière suivante :

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

def F(y,t):
   return(-y+t+1)

y0=3
T=np.arange(0,2.2,0.2)
y=odeint(F,y0,T)

plt.plot(T,y,linestyle='dotted',marker='o',linewidth=2,label='Odeint')
plt.plot(T,3*np.exp(-T)+T,linewidth=2,label='Solution exacte')
plt.legend()
plt.show()
```



On remarque que même avec cette discrétisation très grossière, la résolution est très précise. On peut à présent passer à des équations différentielles non linéaires (dont on ne connait pas la solution exacte) et en tracer une solution approchée. La seule difficulté est de changer la fonction F !

Exercice d'application 5. Tracer le graphe de la solution de l'équation différentielle

$$y' = y \times \left(1 - \frac{y}{3}\right) - e^t$$

sur $[0, 1]$ avec comme condition initiale $y(0) = 1$.

Exercice d'application 6. Tracer le graphe de la solution de l'équation différentielle $y' = -\dfrac{y}{1 + y^2}$ sur $[0, 10]$ avec comme condition initiale $y(0) = 1$.

II.2. Équations d'ordre 2

La fonction odeint s'applique encore mais la situation se complique. En effet, il faut transformer l'équation différentielle d'ordre 2 en équation différentielle vectorielle d'ordre 1. Considérons par exemple l'équation $y'' - (1 - y^2)y' + y = 0$ avec comme condition initiale $y(0) = 0.1$ et $y'(0) = 0$. On va alors poser le vecteur :

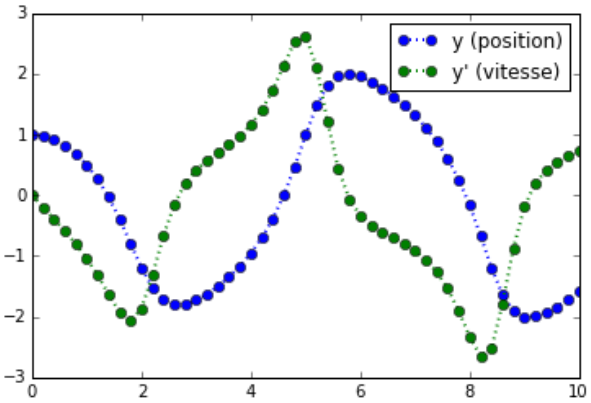
$$Y = \begin{pmatrix} y \\ y' \end{pmatrix}.$$

On a alors en dérivant terme à terme $Y' = \begin{pmatrix} y' \\ y'' \end{pmatrix} = \begin{pmatrix} y' \\ (1-y^2)y' - y \end{pmatrix}$. On a ainsi transformé l'équation en équation du type :

$$Y' = F(Y, t)$$

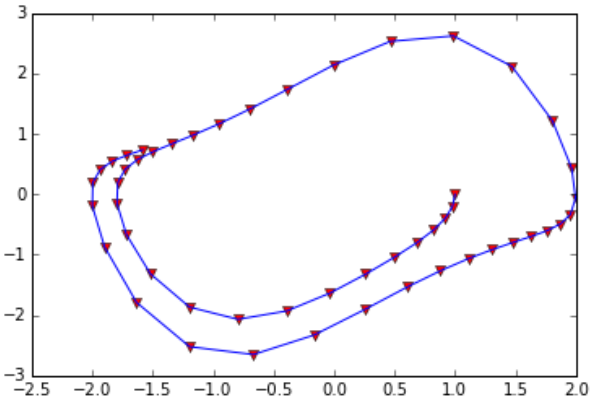
où F est cette fois une fonction qui dépend du vecteur Y et du temps t. Il suffit alors encore une fois de définir la fonction F et d'appliquer odeint qui renverra alors un tableau à 2 colonnes. Dans la première colonne, on aura $[y(t_0), y(t_1), \ldots, y(t_n)]$ et dans la seconde, on aura $[y'(t_0), y'(t_1), \ldots, y'(t_n)]$. On obtient alors par exemple la syntaxe suivante :

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

def F(Y,t):
   return([Y[1], (1.0-Y[0]**2)* Y[1] - Y[0]])

y0=1.0
v0=0.0
T=np.arange(0,10.2,0.2)
y=odeint(F,[y0,v0],T)

plt.plot(T,y[:,0],linestyle='dotted',marker='o',linewidth=2,label='y (position)')
plt.plot(T,y[:,1],linestyle='dotted',marker='o',linewidth=2,label='y\' (vitesse)')
plt.legend()
plt.show()
```



Ce graphe n'a pas vraiment de sens physique (les variables y et y' n'ayant pas les mêmes dimensions). Pour visualiser l'évolution de la position et de la vitesse en même temps, on trace parfois y' en fonction de y. C'est le portrait de phase du système. Ainsi, on a une trajectoire dans le plan où en abscisse on a la position et en ordonnée la vitesse. On part ici du point $(1, 0)$. On voit alors sur le graphe que y commence à diminuer (la vitesse est nulle initialement puis négative), puis la vitesse remonte et quand elle passe 0, notre trajectoire repart vers la droite (la position augmente), etc.

```
plt.plot(y[:,0], y[:,1], 'rv')
plt.plot(y[:,0], y[:,1], 'b-')
plt.legend()
plt.show()
```



Exercice d'application 7. On considère l'équation différentielle $y'' + \sin(y) = 0$ avec condition initiale $y(0) = 1$ et $y'(0) = 0$. *C'est l'équation modélisant l'angle d'un pendule non amorti.*

1) Tracer le graphe de la solution approchée avec odeint sur $[0, 10]$.

2) Comparer le graphe avec celui de la solution exacte de l'équation $y'' + y = 0$ que vous savez résoudre !

3) Tracer le portrait de phase (y' en fonction de y).

Exercice d'application 8. Recommencer l'exercice précédent avec l'équation $y'' + 0.5y' + \sin(y) = 0$ (pendule amorti).

Tout cela ne s'arrête pas ici... On peut aussi avec odeint résoudre des systèmes d'équations différentielles couplées, c'est à dire des équations différentielles de la forme $\begin{cases} x' = F(x, y, t) \\ y' = G(x, y, t) \end{cases}$ *en utilisant la même méthode que ci-dessus et en posant* $Y = \begin{pmatrix} x \\ y \end{pmatrix}$.

III. Correction des exercices

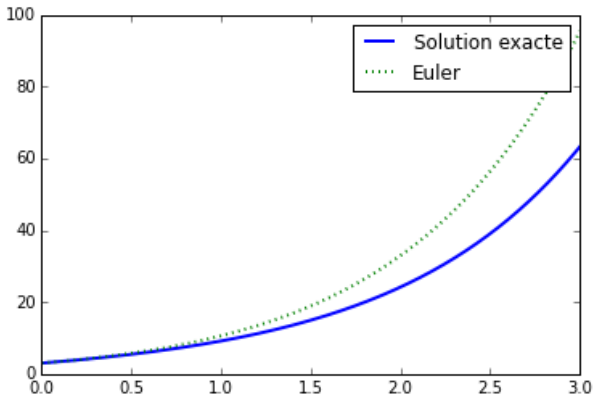
Exercice d'application 1. On a $y' = -y + t + 1$. On a donc $F(y,t) = -y + t + 1$, soit :

```
def F(y,t):
    return(-y+t+1)
```

Exercice d'application 2. $T_3 = np.arange(0, 2.2, 0.2)$.

Exercice d'application 3. Voici le script global et le tracé :

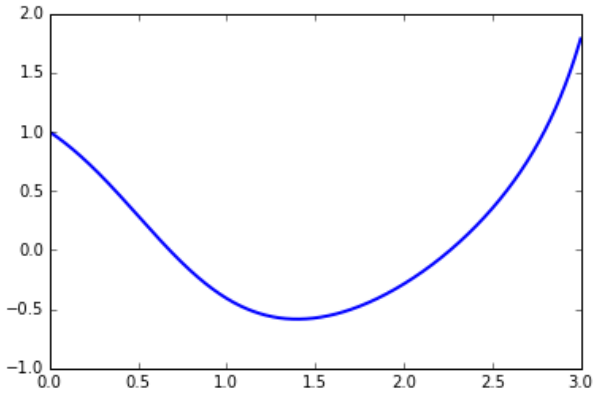
```
import numpy as np
import matplotlib.pyplot as plt

def F(y,t):
    return(y+t+1)

T=np.linspace(0,3,10001)

def Euler(y0,T):
    n=len(T)
    Y=np.zeros(n)
    Y1=np.zeros(n)
    Y[0]=y0
    Y1[0]=F(y0,T[0])
    for k in range(n-1):
        Y[k+1]=Y[k]+(T[k+1]-T[k])*Y1[k]
        Y1[k+1]=F(Y[k+1],T[k+1])
    return(Y)

sol=Euler(3,T)

plt.plot(T,3*np.exp(T)+T,linewidth=2,label='Solution exacte')
plt.plot(T,sol,linewidth=2,linestyle='dotted',label='Euler')
plt.legend()
plt.show()
```



Puisque l'on a une exponentielle croissante, nos erreurs d'approximation sont de plus en plus grandes ce qui fait que notre solution approchée finit par s'éloigner de la solution théorique.

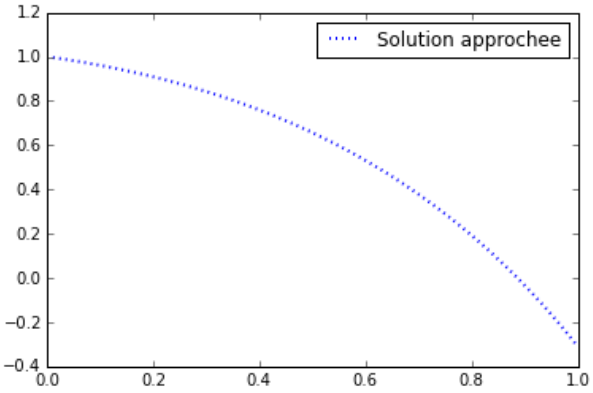
Exercice d'application 4. On reprend le même script en modifiant la fonction F, le tableau T :

```
import numpy as np
import matplotlib.pyplot as plt

def F(y,t):
    return(y**2 -2*np.cos(t))

T=np.linspace(0,3,10001)

def Euler(y0,T):
    n=len(T)
    Y=np.zeros(n)
    Y1=np.zeros(n)
    Y[0]=y0
    Y1[0]=F(y0,T[0])
    for k in range(n-1):
        Y[k+1]=Y[k]+(T[k+1]-T[k])*Y1[k]
        Y1[k+1]=F(Y[k+1],T[k+1])
    return(Y)

sol=Euler(1,T)
plt.plot(T,sol,linewidth=2)
plt.show()
```



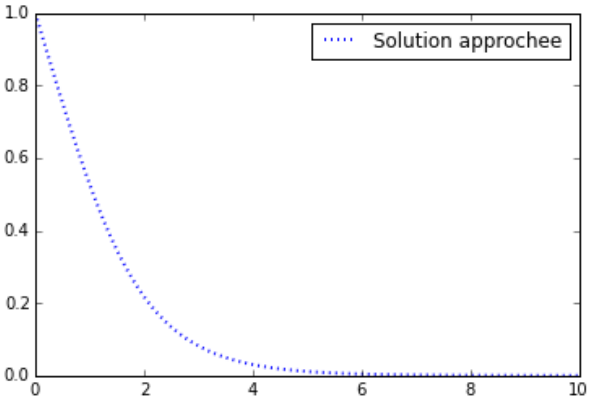
Exercice d'application 5. Pour $y' = y \times \left(1 - \frac{y}{3}\right) - e^t$, on a $F(y,t) = y\left(1 - \frac{y}{3}\right) - e^t$ donc :

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

def F(y,t):
   return(y*(1-y/3)-np.exp(t))

y0=1.0
T=np.arange(0,1.01,0.01)
y=odeint(F,y0,T)

plt.plot(T,y,linestyle='dotted',linewidth=2,label='Solution approchee')
plt.legend()
plt.show()
```



Exercice d'application 6. Pour $y' = -\dfrac{y}{y^2+1}$, on a $F(y,t) = -\dfrac{y}{y^2+1}$. On a donc :

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

def F(y,t):
   return(-y/(y**2+1))

y0=1
T=np.arange(0,10.01,0.01)
y=odeint(F,y0,T)

plt.plot(T,y,linestyle='dotted',linewidth=2,label='Solution approchee')
plt.legend()
plt.show()
```



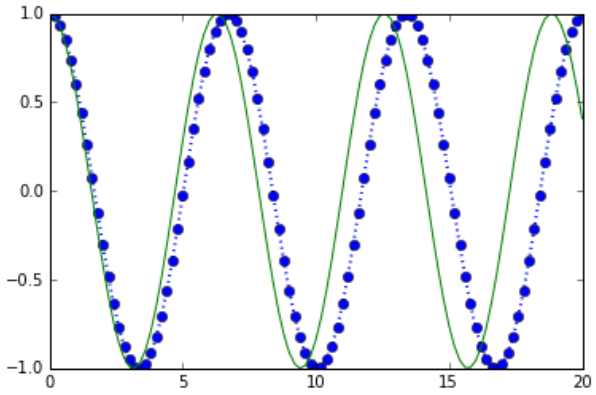
Attention ! La fonction F doit quand même prendre en paramètre t même si son résultat ne dépend pas de t sinon la fonction odeint plantera (elle attend en paramètre une fonction de 2 variables !)

Exercice d'application 7. On pose $Y = \begin{pmatrix} y \\ y' \end{pmatrix}$ donc $Y' = \begin{pmatrix} y' \\ -\sin(y) \end{pmatrix}$. On a donc :

Résoudre l'équation différentielle $y'' + \sin(y) = 0$ avec condition initiale $y(0) = 1$ et $y'(0) = 0$. *On pourra comparer le graphe avec celui de la solution exacte de l'équation $y'' + y = 0$ que vous savez résoudre !*

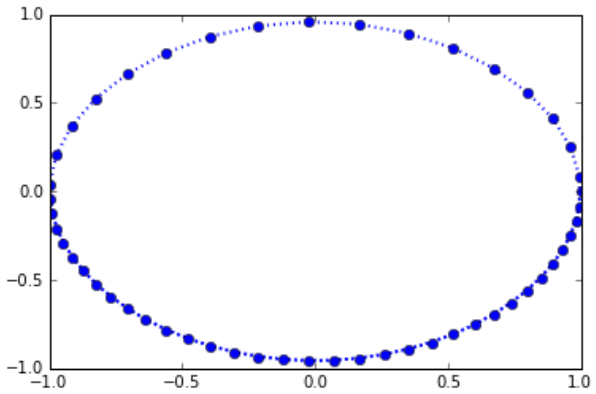
Exercice d'application 8. On répond directement aux questions 1 et 2 (la solution théorique de $y'' + y = 0$ avec $y(0) = 1$ et $y'(0) = 0$ est $y(t) = \cos(t)$). On a tracé en bleu la solution approchée de $y'' + \sin(y) = 0$ et en vert la solution théorique de $y'' + y = 0$.

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

def F(Y,t):
   return([Y[1], -np.sin(Y[0])])

y0=1.0
v0=0.0
T=np.arange(0,10.2,0.2)
y=odeint(F,[y0,v0],T)

plt.plot(T,y[:,0],linestyle='dotted',marker='o',linewidth=2)
plt.plot(T,np.cos(T))
plt.show()
```



On peut alors tracer le portrait de phase :

```
plt.plot(y[:,0],y[:,1],linestyle='dotted',marker='o',linewidth=2)
plt.show()
```

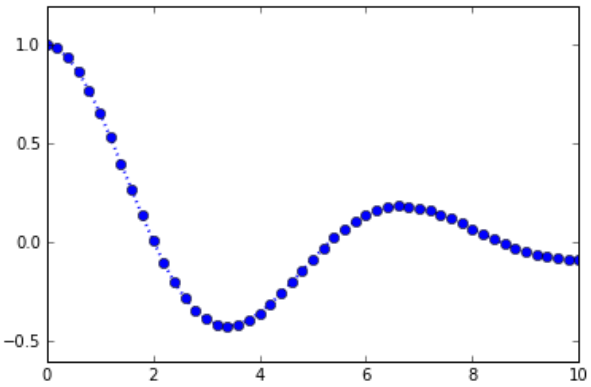


La périodicité de y (et donc le portrait de phase circulaire) n'est pas surprenante car on a étudié ici l'équation d'un pendule non amorti.

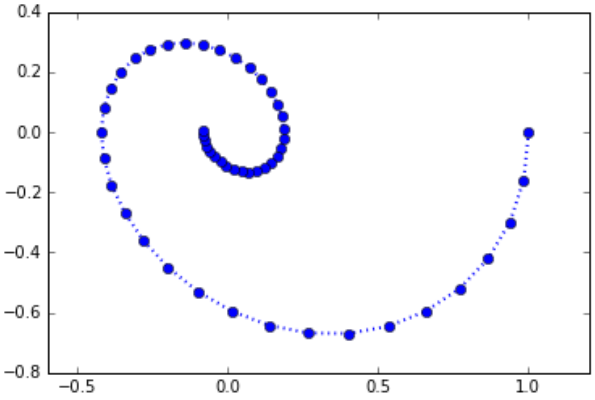
Exercice d'application 9. On a alors :

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

def F(Y,t):
```

```
    return([Y[1], -0.5*Y[1]-np.sin(Y[0])])

y0=1.0
v0=0.0
T=np.arange(0,10.2,0.2)
y=odeint(F,[y0,v0],T)

plt.plot(T,y[:,0],linestyle='dotted',marker='o',linewidth=2)
plt.show()
```




```
plt.plot(y[:,0],y[:,1],linestyle='dotted',marker='o',linewidth=2)
plt.show()
plt.show()
```



On observe alors un amortissement (l'angle y fini par atteindre 0) ce qui est tout à fait normal, le pendule étant amorti !