Résolution numérique d'équations différentielles - Méthode d'Euler

Table des matières

Soit $I = [a; b]$ un intervalle de $\mathbb{R}$ et un réel y_0 . On considère une application $f : I \times \mathbb{R} \longrightarrow \mathbb{R}$ "suffisamment régulière".
Le théorème de Cauchy Lipschitz assure que, sous certaines conditions raisonnables, il existe une unique application y de classe $\mathscr{C}^1$ sur I telle que :

$$\begin{cases} y(a) = y_0 \\ y'(t) = f(t, y(t)) \end{cases} \quad \text{pour tout } t \in I \ .$$

Dans la plupart des cas, on ne sait pas résoudre explicitement le problème de Cauchy . D'où la nécessité de mettre au point des méthodes numériques de résolution approchée d'un tel problème.

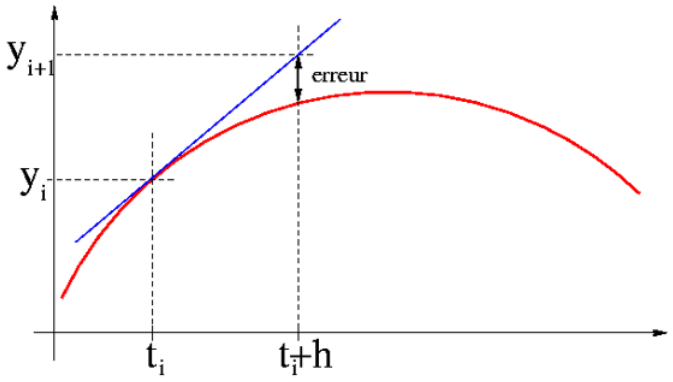
1 La méthode d'Euler.

Principe général : Il s'agit de calculer une approximation des $y(t_i)$ avec $t_i = a + i \times h$ où $h = \frac{b-a}{n}$ et $n \in \mathbb{N}^*$.
Nous savons que $\lim\limits_{h \to 0} \frac{y(t_i+h)-y(t_i)}{h} = y'(t_i) = f(t_i, y(t_i))$. D'où :

$$y(t_{i+1}) \approx y(t_i) + h \times f(t_i, y(t_i))$$

Nous allons noter y_i une approximation de $y(t_i)$ pour tout $i \in [\![0; n]\!]$.
Comment interpréter graphiquement les valeurs ?

- On part du point (a, y_0) , on suit alors la droite de pente $f(a, y_0)$ sur l'intervalle $[a; a + h]$. On pose alors $\begin{cases} t_1 = t_0 + h = a + h \\ y_1 = y_0 + h \times f(t_0, y(t_0)) \end{cases}$
- Partant du point (t_1, y_1) , on suit la droite de pente $f(t_1, y_1))$ sur l'intervalle $[t_1; t_1 + h]$.
- Ainsi de suite...



On construit ainsi une suite de points de la manière suivante :

$$\begin{cases} t_{i+1} = t_i + h \\ y_{i+1} = y_i + h \times f(t_i, y(t_i)) \end{cases} .$$

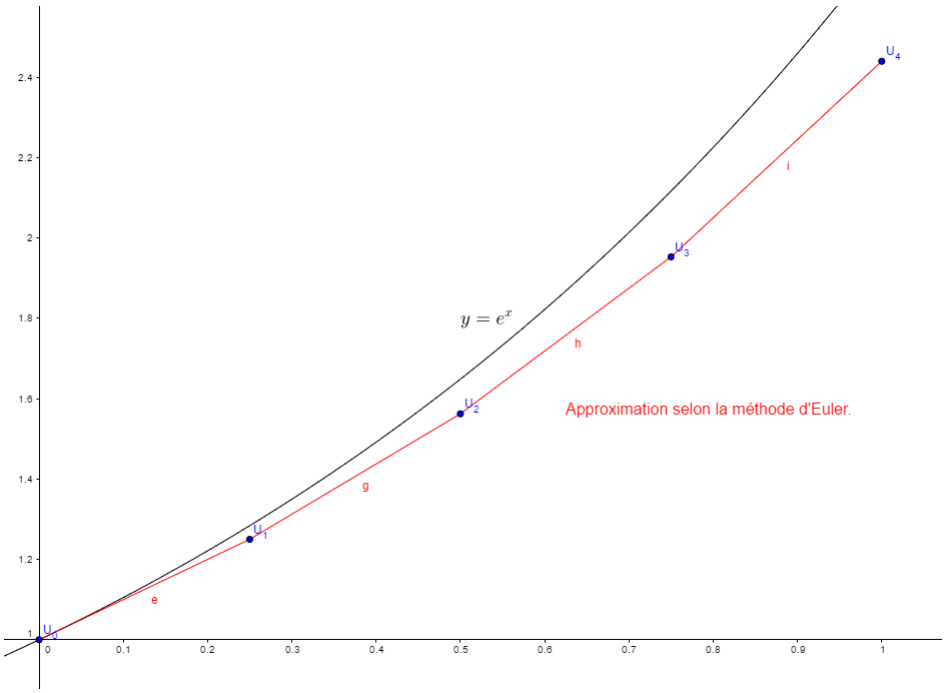
La ligne brisée joignant les points $\{U_i(t_i, y_i), i \in [\![0; n]\!]\}$ constitue une approximation de la courbe de la solution exacte de notre équation différentielle.

Exemple 1.1. *: considérons le problème de Cauchy suivant :* $\begin{cases} y(0) = 1 \\ y'(t) = y(t) \end{cases}$ *sur* $[0; 1]$. *Choisissons* $n = 4$.
Calculons les coordonnées des sommets de la ligne brisée puis traçons cette ligne.
$h = \frac{1-0}{4} = 0,25$ *et on obtient les points :*

- $(t_0, y_0) = (0,1)$
- $t_1 = t_0 + 0,25 = 0,25$ et $y_1 = y_0 + 0.25y_0 = 1,25$
- $t_2 = t_1 + 0,25 = 0,5$ et $y_2 = y_1 + 0,25y_1 = 1,25 + 0,25 \times 1,25 = 1,5625$
- $t_3 = t_2 + 0,25 = 0,75$ et $y_3 = y_2 + 0,25y_2 = 1,5625 + 0,25 \times 1,5625 = 1,953125$
- $t_4 = t_3 + 0,25 = 1$ et $y_4 = y_3 + 0,25y_3 = 1,953125 + 0,25 \times 1,953125 \approx 2,44$

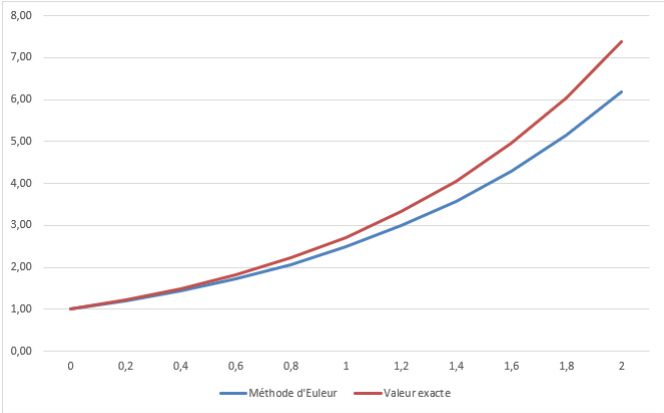
Mais nous savons que la solution exacte à ce problème de Cauchy est la fonction $y = e^x$. *On peut donc calculer les pourcentages d'erreur :*

tk	yk	exp(tk)	% erreur
0	1,00	1,00	0,00
0,25	1,25	1,28	2,65
0,5	1,56	1,65	5,23
0,75	1,95	2,12	7,74
1	2,44	2,72	10,19



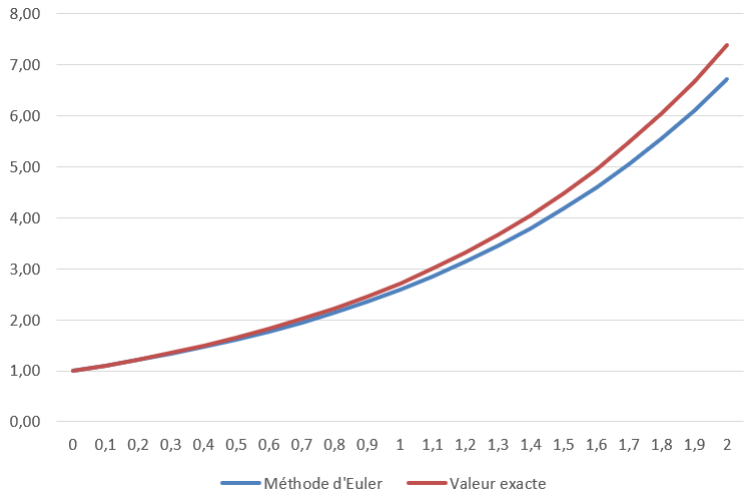
Intéressons-nous à l'intervalle $[0; 2]$ *avec* $n = 10$ *:*

tk	yk	exp(tk)	% erreur
0	1	1	0,0
0,2	1,2	1,22	1,6
0,4	1,44	1,49	3,4
0,6	1,73	1,82	4,9
0,8	2,07	2,23	7,2
1	2,49	2,72	8,5
1,2	2,99	3,32	9,9
1,4	3,58	4,06	11,8
1,6	4,3	4,95	13,1
1,8	5,16	6,05	14,7
2	6,19	7,39	16,2



Intéressons-nous maintenant à l'intervalle $[0;2]$ *avec* $n = 20$ *:*

tk	yk	exp(tk)	% erreur
0	1	1	0,0
0,1	1,1	1,11	0,9
0,2	1,21	1,22	0,8
0,3	1,33	1,35	1,5
0,4	1,46	1,49	2,0
0,5	1,61	1,65	2,4
0,6	1,77	1,82	2,7
0,7	1,95	2,01	3,0
0,8	2,14	2,23	4,0
0,9	2,36	2,46	4,1
1	2,59	2,72	4,8
1,1	2,85	3	5,0
1,2	3,14	3,32	5,4
1,3	3,45	3,67	6,0
1,4	3,8	4,06	6,4
1,5	4,18	4,48	6,7
1,6	4,59	4,95	7,3
1,7	5,05	5,47	7,7
1,8	5,56	6,05	8,1
1,9	6,12	6,69	8,5
2	6,73	7,39	8,9



2 Qualité de la méthode

Pour juger de la qualité d'une méthode d'approximation numérique, il faut prendre en compte plusieurs critères :

- L'**erreur de consistance** : il s'agit d'un ordre de grandeur de l'erreur commise à chaque itération.
  Ici, il s'agit de mesurer l'erreur commise en approximant $y(t_{i+1}) - y(t_i)$ par $h \times f(t_i, y(t_i))$, c'est-à-dire en écrivant

$$y(t_{i+1}) - y(t_i) \approx h \times f(t_i, y(t_i)).$$

C'est l'erreur commise en remplaçant le taux d'accroissement de la fonction en t_i par le nombre dérivé.
Grâce à une formule mathématique (formule de Taylor Lagrange), on peut montrer que cette erreur est dominée par $h^2 = \frac{(b-a)^2}{n^2}$.
Ainsi, plus le pas est petit (n grand), meilleure sera l'approximation.

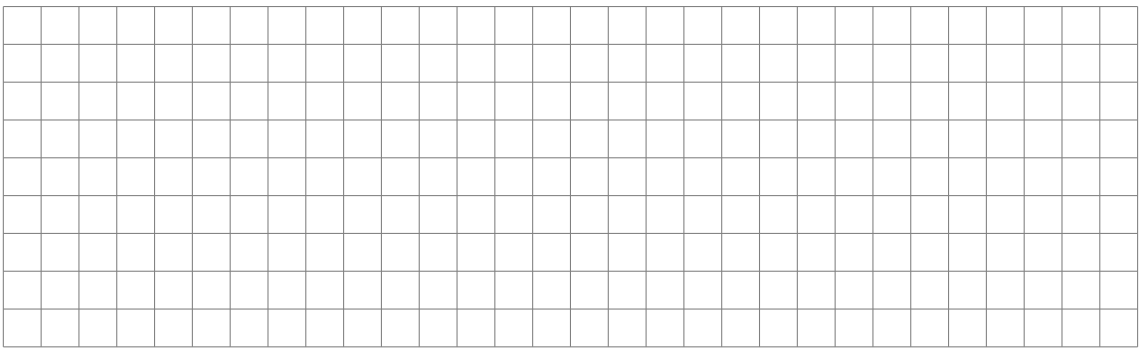
- Le schéma est dit **convergent** lorsque l'erreur globale (qui est le maximum des écarts entre la solution exacte et la solution approchée) tend vers 0 lorsque le pas tend vers 0.
- Les **erreurs d'arrondis** : afin de diminuer l'erreur globale théorique, nous pourrions être tentés d'appliquer la méthode d'Euler avec un pas très petit. Mais nous savons que sous Python, pour des flottants de l'ordre de 10^{-6} , les calculs ne sont plus du tout exacts !

  En pratique, il faut donc choisir h assez petit (pour que la méthode converge rapidement d'un point de vue théorique), mais pas trop petit non plus (pour que les erreurs d'arrondis ne donnent pas lieu à des résultats aberrants et que le temps de calcul ne soit pas trop long).

3 Implémentation sous Python

Exercice 3.1. $\begin{cases} y(1) = 1 \\ ty'(t) - 2y(t) = 0 \end{cases}$ *sur* $[1; 2]$.

1. *Quelle est la solution exacte ?*
2. *Implémenter un script sous Python permettant :*
   - *de tracer la courbe de la solution approchée.*
   - *d'afficher le pourcentage d'erreur pour l'image de 2.*
     Quel est le pourcentage d'erreur par la méthode d'Euler pour l'image de 2 *si* $n = 10$ *? si* $n = 20$ *? si* $n = 100$ *?*



Exercice 3.2. $\begin{cases} y(1) = 0 \\ y'(t) - \frac{3y(t)}{t} - t = 0 \end{cases}$ *sur* $[1;2]$.

1. *Quelle est la solution exacte ?*
2. *Implémenter un script sous Python permettant :*
   - *de tracer la courbe de la solution approchée.*
   - *d'afficher le pourcentage d'erreur pour l'image de 1,5.*
     Quel est le pourcentage d'erreur par la méthode d'Euler pour l'image de 2 *si* $n = 10$ *? si* $n = 20$ *? si* $n = 100$ *?*
3. *Mêmes questions pour l'image de 2.*
4. *Quelles conclusions en tirer ?*

4 Système d'équations différentielles

On veut maintenant résoudre un système d'équations différentielles sur $[a, b]$ (avec plusieurs fonctions inconnues y, z) :

$$\begin{cases} y'(t) = 4y(t) - 2z(t) \\ z'(t) = y(t) + 3z(t) \end{cases}$$

Comment approximer $y(t)$ et $z(t)$?

On peut aussi vectorialiser en posant $Y(t) = \begin{pmatrix} y(t) \\ z(t) \end{pmatrix}$.

$$Y'(t) = \begin{pmatrix} y'(t) \\ z'(t) \end{pmatrix} = \begin{pmatrix} 4 & -2 \\ 1 & 3 \end{pmatrix} Y(t)$$

On est revenu à une E.D. du 1er ordre donc on peut appliquer la méthode d'Euler !
Soit $F : U \subseteq \mathbb{R} \longrightarrow \mathbb{R}^n C^1$, on considère le problème suivant, où $Y(t) \in \mathbb{R}^n, \forall t$

$$\begin{cases} Y'(t) = F(t, Y(t)) \\ Y(t_0) = Y_0 \end{cases}$$

La **méthode d'Euler vectorielle**, de pas h, consiste à approximer la solution Y par une suite $(Y_k)_{0 \leq k < n}$ de vecteurs de $\mathbb{R}^n$ telle que :

- $Y_0 = Y(t_0)$
- $Y_{k+1} = Y_k + h \times F(t_k, Y_k)$

$$Y'(t) = \begin{pmatrix} y'(t) \\ z'(t) \end{pmatrix} = \begin{pmatrix} 4 & -2 \\ 1 & 3 \end{pmatrix} Y(t)$$

Équation de récurrence de la méthode d'Euler vectorielle, avec $Y_k = \begin{pmatrix} y_k \\ z_k \end{pmatrix}$:

$$Y_{k+1} = Y_k + h \times \begin{pmatrix} 4 & -2 \\ 1 & 3 \end{pmatrix} Y_k$$

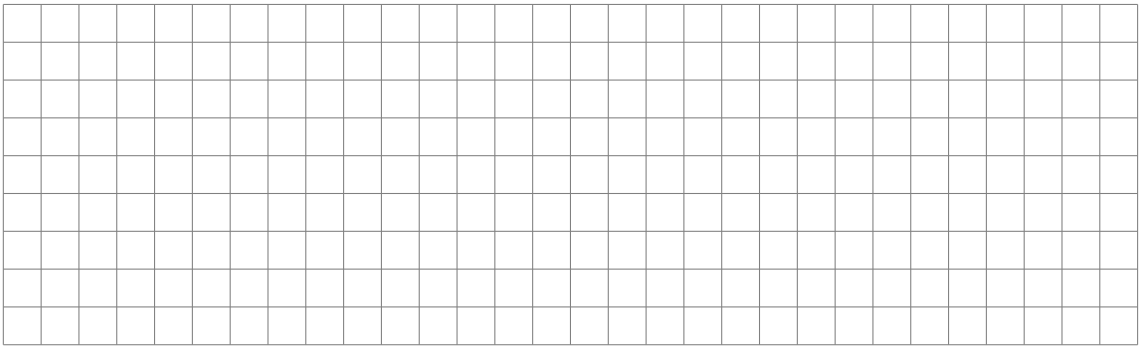
Écrit différemment, avec $Y_k = \begin{pmatrix} y_k \\ z_k \end{pmatrix}$:

$$\begin{cases} y_{k+1} = y_k + h \times (4y_k - 2z_k) \\ z_{k+1} = z_k + h \times (y_k + 3z_k) \end{cases}$$

(ce qui revient à appliquer la méthode d'Euler sur les deux équations différentielles).

Application 4.1. *En déduire un algorithme en Python pour approximer les solutions sur* $[0,1]$ *(avec un pas de 0.1) de :*

$$\begin{cases} y'(t) = 4y(t) - 2z(t) \\ z'(t) = y(t) + 3z(t) \\ y(0) = 3 \\ z(0) = 7 \end{cases}$$



Exemple 4.2. *Soient deux réactions chimiques d'ordre 1 :*

$$A \xrightarrow{\alpha} B$$
$$B \xrightarrow{\beta} C$$

On veut connaitre les concentrations au cours du temps.

$$\begin{cases} \frac{d[A]}{dt} = -\alpha[A] \\ \frac{d[B]}{dt} = \alpha[A] - \beta[B] \\ \frac{d[C]}{dt} = \beta[B] \end{cases}$$

Écrire en Python la méthode d'Euler pour $t \in [0,5]$*, avec* $[A]_0 = 1$ *et*

$$[B]_0 = [C]_0 = 0$$

5 Ordre 2

Comment appliquer la méthode d'Euler sur une équation différentielle d'ordre ≥ 2 ?

Par exemple (équation du pendule linéarisé) :

$$\theta''(t) = -\theta(t)$$

On pose $z(t) = \theta'(t)$.

On a alors $z'(t) = \theta''(t) = -\theta(t)$.

On obtient donc un système d'équations différentielles linéaires d'ordre 1 :

$$\begin{cases} \theta'(t) = z(t) \\ z'(t) = -\theta(t) \end{cases}$$

On peut vectorialiser en posant $\Theta(t) = \begin{pmatrix} \theta(t) \\ \theta'(t) \end{pmatrix}$.

Alors $\Theta'(t) = \begin{pmatrix} \theta'(t) \\ \theta''(t) \end{pmatrix} = \begin{pmatrix} \theta'(t) \\ -\theta(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} \theta(t) \\ \theta'(t) \end{pmatrix}$.

On s'est ramené à une équation différentielle du 1er ordre :

$$\Theta'(t) = A\Theta(t).$$

Les approximations de la méthode d'Euler sont donc des vecteurs Θ_k vérifiant :

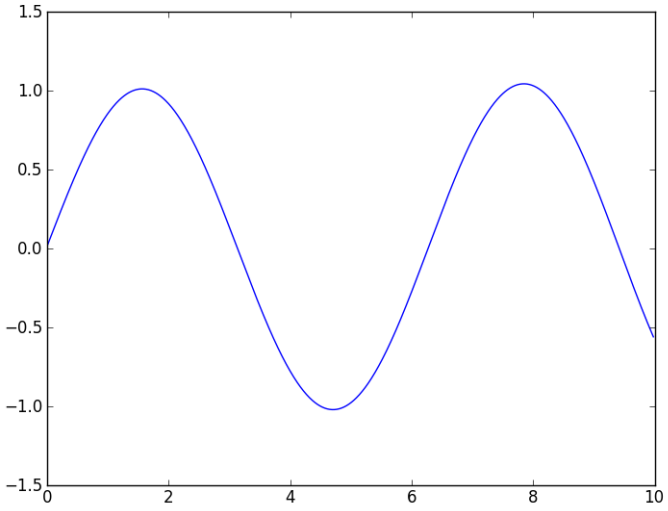
$$\Theta_{k+1} = \Theta_k + h \times \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \Theta_k$$

Voici le script Python correspondant pour $\theta(0) = 0$, $\theta'(0) = 1$ sur $[0; 10]$ avec $n = 1000$ subdivisions :

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


theta=[0]
theta_p=[1]
T=[0]
for k in range(999):
    theta.append(theta[k]+0.01*theta_p[k])
    theta_p.append(theta_p[k]-0.01*np.sin(theta[k]))
    T.append(k*0.01)
plt.plot(T,theta)
plt.show()
```

La courbe obtenue est la suivante :



Équation du pendule non linéarisé :

$$\theta''(t) = -\sin(\theta(t))$$

On pose $\Theta(t) = \begin{pmatrix} \theta(t) \\ \theta'(t) \end{pmatrix}$.

Alors $\Theta'(t) = \begin{pmatrix} \theta'(t) \\ \theta''(t) \end{pmatrix} = \begin{pmatrix} \theta'(t) \\ -\sin(\theta(t)) \end{pmatrix}$.

Application 5.1. *Rédiger le script Python pour l'équation non linéarisée avec les mêmes conditions initiales.*

Application 5.2. *Afficher la courbe représentation d'une approximation de la solution du problème de Cauchy suivant sur* $[0;5]$ *:* $\begin{cases} y'' + xy' + e^x y = 0 \\ y(0) = 1 \\ y'(0) = 0 \end{cases}$

6 Utilisation de bibliothèques Python

6.1 Équation différentielle d'ordre 1

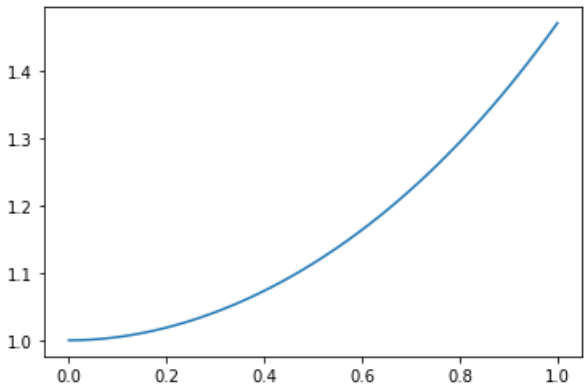
Pour résoudre une équation différentielle $x' = f(x,t)$, on peut utiliser la fonction ***odeint*** du module ***scipy.integrate***.
Cette fonction nécessite une liste de valeurs de t, commençant en t_0, et une condition initiale x_0.
La fonction renvoie des valeurs approchées (aux points contenus dans la liste des valeurs de t) de la solution x de l'équation différentielle qui vérifie $x(t_0) = x_0$.
Pour trouver des valeurs approchées sur $[0,1]$ de la solution de l'équation différentielle :

$$y'(t) = ln(t+1)y(t)$$

qui vérifie $x(0) = 1$, on peut employer le code suivant :

```
1  import numpy as np
2 import scipy.optimize as resol
3 import scipy.integrate as integr
4 import matplotlib.pyplot as plt
5
6
7 def f(y, t) :
8     return np.log(t+1)*y
9
10 T = np.arange(0, 1.01, 0.01)
11 X = integr.odeint(f, 1, T)
12 plt.plot(T,X)
13 plt.show()
14
```

On obtient la courbe :




```
In [14]: X[0]
Out[14]: array([1.])

In [15]: X[-1]
Out[15]: array([1.47151778])
```

6.2 Système différentiel

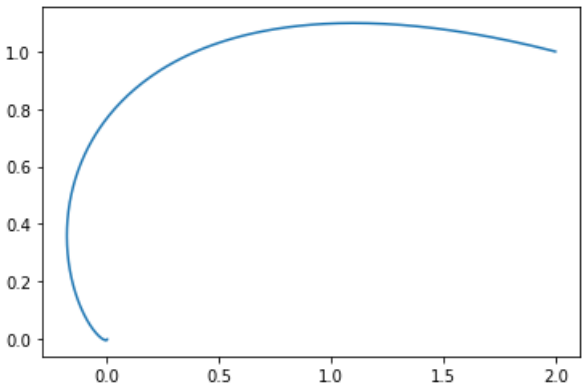
Si on souhaite résoudre, sur $[0, 1]$, le système différentiel :

$$\begin{cases} x'(t) = -2 * x(t) - y(t) \\ y'(t) = x(t) - y(t) \end{cases}$$

avec la condition initiale $x(0) = 2, y(0) = 1$ le code devient le suivant ;

```
1  import numpy as np
2  import scipy.optimize as resol
3  import scipy.integrate as integr
4  import matplotlib.pyplot as plt
5
6  def f(x, t) :
7      return np.array([-2*x[0]-x[1], x[0]-x[1]])
8
9  T = np.arange(0, 5.01, 0.01)
10 X = integr.odeint(f, np.array([2.,1.]), T)
11 plt.plot(X[ :,0], X[ :,1])
12 plt.show()
```

On obtient la courbe :




```
In [18]: X[0]
Out[18]: array([2., 1.])

In [19]: X[-1]
Out[19]: array([ 0.00077248, -0.00168769])
```

6.3 Équation différentielle d'ordre 2

Pour résoudre une équation différentielle scalaire d'ordre 2 de solution x, on demandera la résolution du système différentiel d'ordre 1 satisfait par $X(t) = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}$.

Ainsi, si on considère la fonction x qui vérifie l'équation différentielle

$$x''(t) + 2x'(t) + 4x(t) = \cos(t)$$

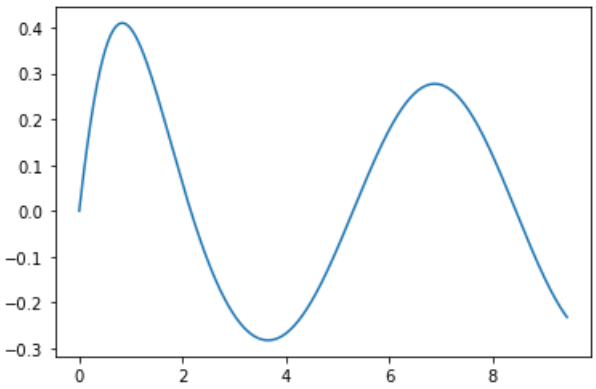
avec les conditions initiales $x(0) = 0, x'(0) = 1$ et , le vecteur X vérifiera

$$X'(t) = \begin{pmatrix} x'(t) \\ -2x'(t) - 4x(t) + \cos(t) \end{pmatrix}.$$

Pour obtenir la représentation graphique de x sur l'intervalle $[0, 3\pi]$, on pourra utiliser le code suivant :

```
1  import numpy as np
2  import scipy.optimize as resol
3  import scipy.integrate as integr
4  import matplotlib.pyplot as plt
5
6   def f(x,t) :
7      return np.array([x[1], -2*x[1] - 4*x[0] + np.cos(t)])
8
9  T = np.arange(0, 3*np.pi + 0.01, 0.01)
10 X = integr.odeint(f, np.array([0,1]), T)
11 plt.plot(T, X[ :,0])
12 plt.show()
13
```

On obtient :


0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
2
4
6
8