

**Informatique en CPGE (2018-2019)**  
**Résolution numérique d'équations**  
**différentielles : méthode d'Euler**

Les équations différentielles permettent de modéliser de nombreux phénomènes physiques. En général, on ne dispose pas de solutions analytiques : par exemple, l'équation  $\theta'' = -k_1 \sin \theta - k_2 \theta'$  permet d'étudier le mouvement d'un pendule amorti et il donc est intéressant de pouvoir visualiser une approximation de la solution.

## 1 Méthode d'Euler

S'il existe une unique solution  $y$ , sur un intervalle  $[a; b]$ , de l'équation  $y'(x) = f(x, y(x))$  avec  $y(a)$  fixé, il s'agit d'approcher  $y$  en un certain nombre de points répartis dans cet intervalle. En particulier, si  $n + 1$  points sont répartis régulièrement sur  $[a; b]$ , on définit le pas  $h = \frac{b-a}{n}$ , soit  $x_k = a + kh$  pour  $k = 0, 1, 2, \dots, n$ . L'objectif est de calculer des approximations  $y_k$  des valeurs  $y(x_k)$ .

On utilise l'approximation  $\frac{y(x+h) - y(x)}{h} \simeq y'(x)$  appliquée pour chaque  $x_k$ , et on obtient

$$y(x_{k+1}) - y(x_k) \simeq hy'(x_k) = hf(x_k, y(x_k)) \simeq hf(x_k, y_k)$$

**Schéma :** on calcule les approximations pour  $k = 0, 1, 2, \dots, n - 1$  par :

$$x_{k+1} = x_k + h \text{ et } y_{k+1} = y_k + hf(x_k, y_k)$$

On initialise avec  $y_0 = y(x_0) = y(a)$  (qui est la seule valeur exacte).

Une programmation de ce schéma consiste à construire deux listes, une pour la suite  $(x_k)$  des abscisses et une pour la suite  $(y_k)$  des ordonnées. On définit une fonction `euler` qui prend en arguments les valeurs extrêmes de l'intervalle  $a$  et  $b$ , la valeur initiale  $y(0)$ , le pas  $h$ , la fonction  $f$  et renvoie les listes des abscisses et des ordonnées.

```
def euler(a, b, y0, h, f):  
    y = y0  
    x = a  
    liste_y = [y0] # la liste des ordonnées renvoyées  
    liste_x = [a] # la liste des abscisses renvoyées  
    while x + h <= b:  
        y += h*f(x, y)  
        liste_y.append(y)  
        x += h  
        liste_x.append(x)  
    return liste_x, liste_y
```

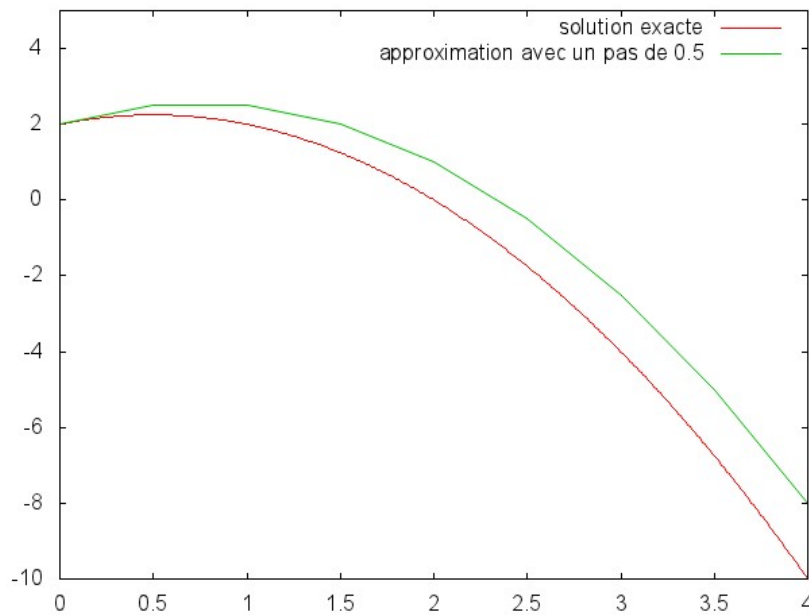
## 2 Exemples

### 2.1 Exemple 1

On cherche une solution approchée de l'équation différentielle  $y' = -2x + 1$ , avec  $y(0) = 2$ , sur l'intervalle  $[0; 4]$ . La solution exacte est  $y(x) = -x^2 + x + 2$ .

Avec la méthode d'Euler, on calcule  $y_{k+1} = y_k + hf(x_k, y_k)$ .

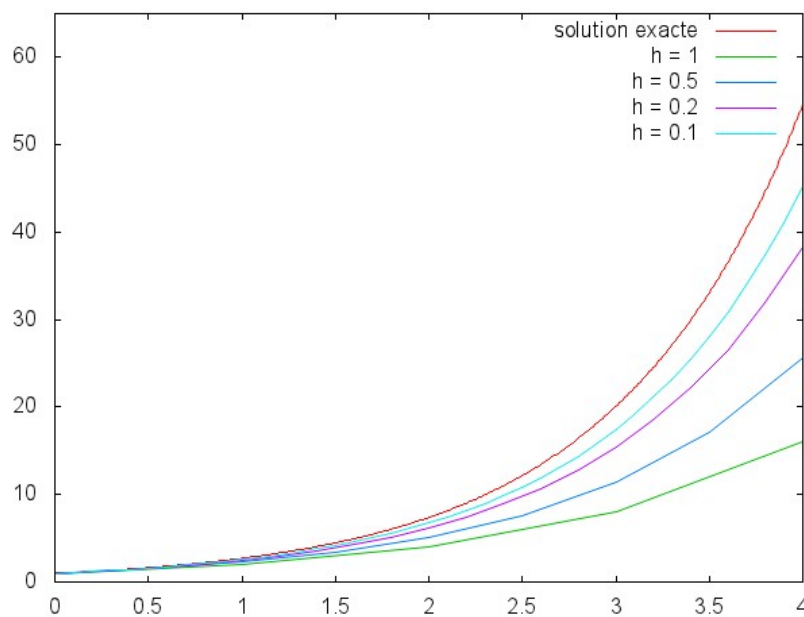
On obtient la figure suivante avec un pas  $h = 0.5$  :



## 2.2 Exemple 2

On cherche une solution approchée de l'équation  $y' = y$ , avec  $y(0) = 1$ , sur l'intervalle  $[0; 5]$ . La solution exacte est  $y(x) = e^x$ . Avec la méthode d'Euler, on calcule  $y_{k+1} = y_k + hy_k$ , soit  $y_{k+1} = (1+h)y_k$ . Avec  $y_0 = 1$ , on obtient  $y_{k+1} = (1+h)^{k+1}$ .

La figure suivante est réalisée avec différentes valeurs du pas  $h$ .



Avec la méthode d'Euler, l'erreur a deux causes constatées sur les exemples précédents : des erreurs

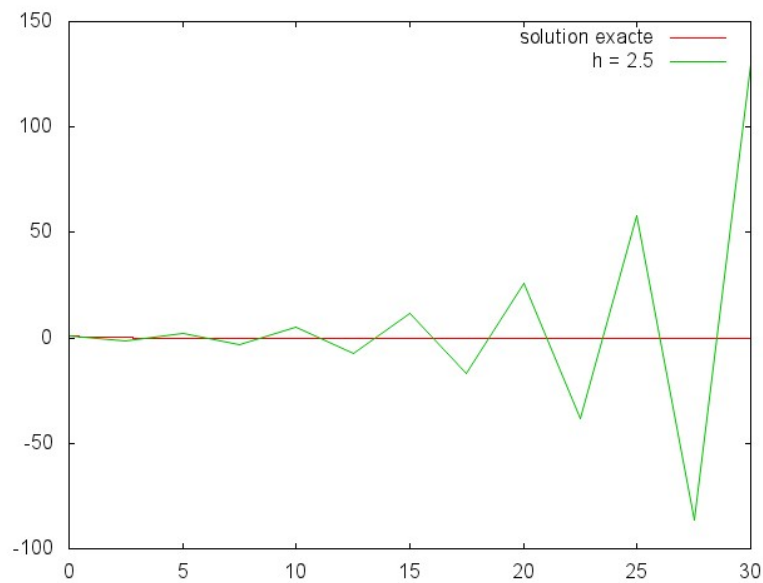
d'arrondi dans les opérations effectuées par l'ordinateur et une erreur de discrétisation, ( $e_k = y_k - y(x_k)$ ), due au procédé de calcul.

Il est important que l'erreur de discrétisation diminue lorsque le pas  $h$  diminue. On dit que la méthode converge si, pour tout  $k$ ,  $y_k$  tend vers  $y(x_k)$  quand  $h$  tend vers 0. Dans ce cas il faudra comme souvent faire un compromis entre la précision de l'approximation et le temps de calcul.

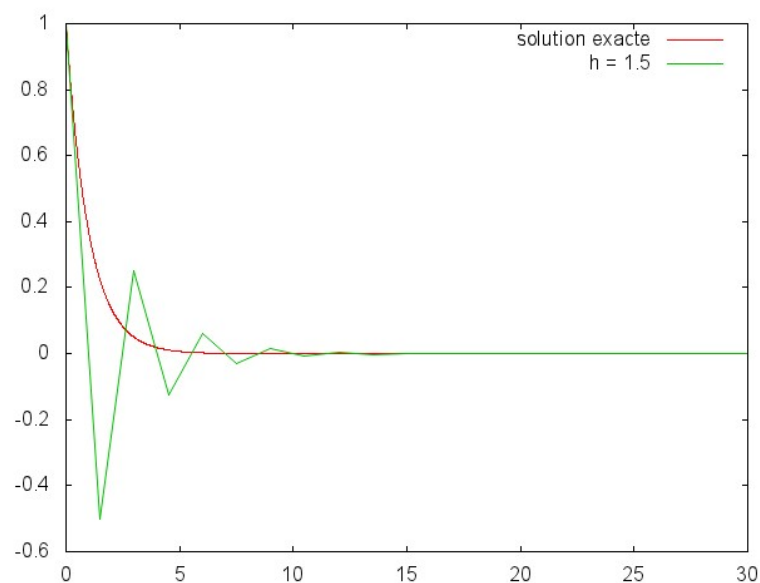
### 2.3 Exemple 3

Problème de stabilité : on résout l'équation  $y' = -y$  avec  $y(0) = 1$  sur l'intervalle  $[0; 30]$ . La solution exacte est  $y(x) = e^{-x}$ . Ici l'intervalle est "grand" et si le pas n'est pas assez petit, on a un problème de stabilité.

Instabilité pour  $h = 2,5$  :



Stabilité pour  $h = 1,5$  :



### 3 Complément

On peut résoudre une équation différentielle de degré 2 ou plus en vectorisant l'équation.

L'équation  $y'' + y = 0$  est équivalente à  $(y, y')' = (y', -y) = F(y, y')$ , soit en posant  $Y = (y, y')$ , on obtient l'équation  $Y' = F(Y)$ . De manière générale, on résout  $Y' = F(x, Y)$ .

La méthode d'Euler peut s'appliquer ici et pour la programmation,  $Y$  sera un objet de type **tuple** ou **list**. Mais les calculs se compliquent car par exemple :  $(3,4)+(2,5)=(3,4,2,5)$  (concaténation des deux couples) et non pas  $(5,9)$  comme on le souhaiterait. Il faudra donc en particulier détailler le calcul de  $Y + hF(Y)$  dans la définition de la fonction `euler`.

On commence par modifier la définition de la fonction `f` pour l'exemple donné :

```
def f(x,y): # y est un couple
    return (y[1],-y[0])
```

Puis la définition de la fonction `euler` :

```
def euler(a, b, y0, h, f):
    x = a
    y = y0 # y0 est un couple
    liste_x = [a]
    liste_y = [y0] # liste de couples
    while x + h <= b:
        y = [y[0] + h * (f(x, y)[0]), y[1] + h * (f(x, y)[1])] #difficulté
        liste_y.append(y)
        x += h
        liste_x.append(x)
    return liste_x, liste_y
```

Il est aussi possible et plus simple d'utiliser un objet de type **array**, (tableau en français).

Un objet de type **array** ressemble à un objet de type **list**, mais ici, tous les éléments doivent être du même type et le nombre d'éléments doit être connu à la création. Les objets de type **array** se trouvent dans une bibliothèque appelée "Numerical Python" (**NumPy**) élaborée pour un calcul numérique optimisé.

Il est alors plus simple de calculer avec des tableaux car les opérations mathématiques sont prédéfinies.

Exemples d'utilisation :

```
import numpy as np
a = np.array([3, 4]) # convertit une liste en tableau
b = np.array([2, 5])
print(a + b) # affiche [5 9]
print(3 * a) # affiche [9 12]
print((a + b)[0]) # affiche 5
```

Maintenant, il n'est plus nécessaire de modifier la définition de la fonction `euler`.

```
def euler(a, b, y0, h, f):
    x = a
    y = y0
    liste_x = [a]
    liste_y = [y0]
    while x + h <= b:
        y = y + h * f(x, y)    # calcul sur les tableaux
        liste_y.append(y)
        x += h
        liste_x.append(x)
    return liste_x, liste_y
```

Il faut cependant modifier la définition de la fonction  $f$  qui renvoie un objet de type **array** et l'appel de la fonction **euler** :

```
def f(x, y):
    return np.array((y[1], -y[0])) # y est un array (F(a, b)=(b, -a))

x, y = euler(a, b, np.array((0, 1)), h, f) #initialisation y(0)=0 et y'(0)=1
```

### Utilisation de la bibliothèque Scipy.

Exemple d'utilisation de la fonction `odeint` de `scipy.integrate`.

```
from math import cos, sin, pi
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integ

def f(x, t): # attention à l'ordre: x'(t) = f(x(t), t)
    return 2 * (cos(t * x)) * x * (1-x/2)
t = np.linspace(0, 15, num=300)
sol = integ.odeint(f, 4, t)

plt.grid()
plt.plot(t, sol)
plt.show()
```

Exemple du pendule pesant amorti :

```
def f(u, t):
    return [u[1], 10 * sin(u[0]) - u[1]/4]

t = np.linspace(0, 10, num=200)
sol = integ.odeint(f, [pi/2, 0], t)

plt.subplot(2, 1, 1)
plt.grid()
plt.plot(t, sol[:,0]) # angle fonction de t
plt.subplot(2, 1, 2)
plt.grid()
plt.plot(sol[:,0], sol[:,1]) # diagramme de phase
plt.show()
```